Q63-6N

MOBILE
A MOBIDIC
COBOL COMPILER

3rd QUARTERLY
PROGRESS REPORT
CONTRACT NO.
DA-36-039-SC-89231

1 OCTOBER 1962
to
1 JANUARY 1963

SYLVANIA ELECTRONIC SYSTEMS
Government Systems Management
for GENERAL TELEPHONE & ELECTRONICS

AD____________________ Accession No.__________

Sylvania Electronic Systems–East, Sylvania Electronic Systems, A Division of Sylvania Electric Products, Inc., Needham Heights 94, Massachusetts.

Report No. Q63-6N, 1 January 1963, 91 pp., 4 ill.
Contract No. DA-36-039-sc-89231

UNCLASSIFIED

Progress is reported for work accomplished on the table formats and functions internal to MOBILE with regards to MUST, Run 4 Output Table Formats, Data Design Table, Run 8 Table Formats, Macro-Instructions and Related Table Formats, COBOL Compiler Output Listings, and Qualification Task in Run 1.3 and a description of the Data-Name List.

UNCLASSIFIED

1. **MOBILE, A MOBIDIC COBOL COMPILER**
2. **Contract No. DA-36-039-sc-89231**

AD____________________ Accession No.__________

Sylvania Electronic Systems East, Sylvania Electronic Systems, A Division of Sylvania Electric Products, Inc., Needham Heights 94, Massachusetts.

Report No. Q63-6N, 1 January 1963, 91 pp., 4 ill.
Contract No. DA-36-039-sc-89231

UNCLASSIFIED

Progress is reported for work accomplished on the table formats and functions internal to MOBILE with regards to MUST, Run 4 Output Table Formats, Data Design Table, Run 8 Table Formats, Macro-Instructions and Related Table Formats, COBOL Compiler Output Listings, and Qualification Task in Run 1.3 and a description of the Data-Name List.

UNCLASSIFIED

1. **MOBILE, A MOBIDIC COBOL COMPILER**
2. **Contract No. DA-36-039-sc-89231**

AD____________________ Accession No.__________

Sylvania Electronic Systems–East, Sylvania Electronic Systems, A Division of Sylvania Electric Products, Inc., Needham Heights 94, Massachusetts.

Report No. Q63-6N, 1 January 1963, 91 pp., 4 ill.
Contract No. DA-36-039-sc-89231

UNCLASSIFIED

Progress is reported for work accomplished on the table formats and functions internal to MOBILE with regards to MUST, Run 4 Output Table Formats, Data Design Table, Run 8 Table Formats, Macro-Instructions and Related Table Formats, COBOL Compiler Output Listings, and Qualification Task in Run 1.3 and a description of the Data-Name List.

UNCLASSIFIED

1. **MOBILE, A MOBIDIC COBOL COMPILER**
2. **Contract No. DA-36-039-sc-89231**

AD____________________ Accession No.__________

Sylvania Electronic Systems–East, Sylvania Electronic Systems, A Division of Sylvania Electric Products, Inc., Needham Heights 94, Massachusetts.

Report No. Q63-6N, 1 January 1963, 91 pp., 4 ill.
Contract No. DA-36-039-sc-89231

UNCLASSIFIED

Progress is reported for work accomplished on the table formats and functions internal to MOBILE with regards to MUST, Run 4 Output Table Formats, Data Design Table, Run 8 Table Formats, Macro-Instructions and Related Table Formats, COBOL Compiler Output Listings, and Qualification Task in Run 1.3 and a description of the Data-Name List.

UNCLASSIFIED

1. **MOBILE, A MOBIDIC COBOL COMPILER**
2. **Contract No. DA-36-039-sc-89231**

MOBILE

A MOBIDIC COBOL COMPILER

THIRD QUARTERLY PROGRESS REPORT

1 October 1962 to 1 January 1963

Signal Corps

Technical Requirements

SCL-2101N

Contract No. DA-36-039-sc-89231

Submitted by: [illegible]

E.W. Jervis, Jr. Manager
MOBIDIC Projects

SYLVANIA ELECTRONIC SYSTEMS–EAST

SYLVANIA ELECTRONIC SYSTEMS
A Division of Sylvania Electric Products Inc.
189 B Street–Needham Heights 94, Massachusetts

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

LIST OF ILLUSTRATIONS

SECTION I

PURPOSE

MOBILE, A COBOL COMPILER FOR THE DATA PROCESSING NEEDS OF THE ARMY SIGNAL CORPS

The objective of this procurement is to produce, for MOBIDIC Computer (C, D, 7A) a COBOL Compiler capable of accepting a Source Program written in COmmon Business Oriented Language, and compiling an Object Program capable of being operated on the above computers.

This procurement will result in delivery to the U. S. Army Signal Corps of a COBOL Compiler (MOBILE I) as specifically defined in Required—COBOL 1961 and with certain features as specified in Elective—COBOL 1961. The MOBILE I Task is being implemented by the Applied Programming Department of the Programming and Analysis Laboratory.

SECTION II

ABSTRACT

The program during this period included system test build-up and further modification of MOBILE required to meet the needs manifested by system testing. This report will concern itself chiefly with specific table formats and functions internal to MOBILE. The following topics are discussed herein:

1. MUST (MOBILE Utility Symbolic Translator)
2. Run 4 Output Table Formats
3. Data Design Table (DDT)
4. Run 8 Table Formats
5. Macro Instructions and Related Table Formats
6. COBOL Compiler Output Listings
7. Qualification Task in Run 1.3 and a description of the Data Name List (DNLA).

SECTION III

PUBLICATIONS, LECTURES, REPORTS AND CONFERENCES

3.1 PUBLICATIONS

Title: A Supplement to the COBOL Preliminary Reference Manual

Date: 1 December 1962

3.2 LECTURES

None

3.3 REPORTS

1. Monthly Letter Report 14 November 1962.
2. Monthly Letter Report 7 December 1962.

3.4 CONFERENCES

1.	Date	:	4 October 1962
	Location	:	Fort Monmouth, New Jersey
	Participants	:	Signal Corps, Sylvania
	Subject	:	Overall Schedule Review and Technical Review of Progress to Date
2.	Date	:	12 October 1962 – 23 October 1962
	Location	:	Orleans, France and Zweibrücken, Germany
	Participants	:	Signal Corps, Sylvania
	Subject	:	Discussion of Technical Data Necessary for Testing MOBIDIC COBOL Compiler (MOBIDIC MOBILE I)
3.	Date	:	12 November 1962
	Location	:	Sylvania, Needham, Mass.
	Participants	:	Signal Corps, Sylvania
	Subject	:	Overall Project Status
4.	Date	:	10 December 1962 – 13 December 1962
	Location	:	Sylvania, Needham, Mass.
	Participants	:	Signal Corps, COMZ, and Sylvania
	Subject	:	Overall Technical Discussion of COBOL and the MOBIDIC Compiler

SECTION IV
FACTUAL DATA

4.1 GENERAL

Specific table formats and functions internal to MOBILE are described in detail in this section. The task breakdown is as follows:

. MOBILE Utility Symbolic Translator – MUST

. Run 4 Output Table Formats

. Data Design Table (DDT)

. Run 8 Table Formats

. Instructions and Related Table Formats

. MOBILE Compiler Output Listings

. Qualification Task in Run 1.3 and Data Name List Discussion

4.2 MOBILE UTILITY SYMBOLIC TRANSLATOR – MUST

4.2.1 Introduction

The design of MOBILE includes an intermediate language called CAP. Since all the statements written in the CAP language exist within MOBILE and are intended to remain unknown to the user of MOBILE, the CAP language is as machine-oriented as possible. This requires the creators of the CAP instructions (the compiler programmers) to either "speak CAP" or make use of a translator.

To the creators of the CAP instructions, CAP itself is only an intermediate language, the result of a CAP translation (i.e., binary machine instructions) being the desired end. MODAL* is a language that may be translated (by an assembly program of the same name) into binary machine instructions.

*MODAL is described in the Second Quarterly Report. It is the "OWN CODE" used with the MOBILE ENTER verb. When MODAL statements are processed by MOBILE run 1.4 and the ENTER generator, CAP statements are generated internally.

However, MODAL is much more user-oriented than CAP. It was thought advantageous to combine the user-oriented qualities of MODAL and machine-oriented qualities of CAP.

This was done by creating a new language, MUST* (MOBILE Utility Symbolic Translator). It is currently being debugged and will translate MUST coding into CAP language, constructing a CAP skeleton table for the MOBILE System according to the requirements of Run 6. The characteristic of MUST is that a DUST** translation of it will produce desirable machine code. On the other hand, a MUST translation of it will produce CAP statements which on a subsequent CAP translation will produce the machine code.

Two desirable conditions have been obtained through the use of MUST:

1. CAP instructions are more easily produced, and
2. CAP instructions are translated and debugged as machine instructions without waiting for a CAP-to-binary translator.

Since the potential user of MUST is essentially working backwards from machine code to CAP, and thence to MUST, it is perhaps best to start by describing the CAP language and show how its various elements are obtained in MUST. CAP instructions are described in detail in Section 4. 6. 3.

4. 2. 2 Features of CAP Language

The features of CAP Language that require explanation are the symbolic location, the symbolic operation code, and the segmentation and contents of the variable field.

*MUST is a tool for the compiler programmer used in the development and debugging of generators for incorporation within the compiler.

**DUST (MOBIDIC D UNLIMITED SYMBOL TABLE) as on page 4-1 of the First Quarterly Report.

4.2.2.1 MUST Symbols

The CAP symbolic location is blank in the CAP Skeleton Table except for certain CAP instructions within subroutines. These subroutine CAP instructions have a 15-bit "internal subroutine symbol." (The symbolic location field of a MUST statement will give a corresponding "internal subroutine symbol" in the resulting CAP instruction if it follows the rules for Subroutine Reference Operands.) In all other cases the symbol appearing in the symbolic location field of a MUST statement will be used only as a location symbol, useful in computing the relative addressing required by MOBILE data self-reference operands.

4.2.2.2 MUST Operation Codes

The CAP symbolic operation code has 9 bits and includes the CAP class (see Section 4.6.3.2) and the machine instruction or pseudo code. The correspondence between MUST symbolic operation codes and CAP symbolic operation codes will be given in conjunction with a description of the variable field options.

4.2.2.3 MUST Variable Fields

The CAP variable field contains four basic entities:

1. a binary constant
2. a MOBILE data reference
3. a subroutine reference
4. a zero with the corresponding number (bits 13-15) indicating that an operand will be supplied at compilation time

The CAP variable field may be divided into one, two, or three subfields.

The MUST variable field contains one to three subfields which in general resemble those of MODAL and which will be enumerated under the Operation Codes. However, it should first be understood how MUST will treat each subfield. Each subfield may be composed of one or more elements connected by the arithmetic connectors + or -. The elements may correspond to any of the three

4.2.2.1 MUST Symbols

The CAP symbolic location is blank in the CAP Skeleton Table except for certain CAP instructions within subroutines. These subroutine CAP instructions have a 15-bit "internal subroutine symbol." (The symbolic location field of a MUST statement will give a corresponding "internal subroutine symbol" in the resulting CAP instruction if it follows the rules for Subroutine Reference Operands.) In all other cases the symbol appearing in the symbolic location field of a MUST statement will be used only as a location symbol, useful in computing the relative addressing required by MOBILE data self-reference operands.

4.2.2.2 MUST Operation Codes

The CAP symbolic operation code has 9 bits and includes the CAP class (see Section 4.6.3.2) and the machine instruction or pseudo code. The correspondence between MUST symbolic operation codes and CAP symbolic operation codes will be given in conjunction with a description of the variable field options.

4.2.2.3 MUST Variable Fields

The CAP variable field contains four basic entities:

1. a binary constant
2. a MOBILE data reference
3. a subroutine reference
4. a zero with the corresponding number (bits 13-15) indicating that an operand will be supplied at compilation time

The CAP variable field may be divided into one, two, or three subfields.

The MUST variable field contains one to three subfields which in general resemble those of MODAL and which will be enumerated under the Operation Codes. However, it should first be understood how MUST will treat each subfield. Each subfield may be composed of one or more elements connected by the arithmetic connectors + or -. The elements may correspond to any of the three entities allowed in the CAP variable field. All subfields are evaluated by a standard routine which gives rise to two quantities, a SYMBOL and an INCREMENT. Depending on the symbolic operation code and the particular subfield being translated, one or both of these quantities will be used to create the corresponding CAP instruction subfield. These quantities, SYMBOL and INCREMENT, are now explained in the light of the four entities available in a CAP subfield.

Decimal or octal numbers appearing alone in a MUST subfield will be convertedto binary and added to the INCREMENT. Standard symbols (ACC, QRG, etc., of which a list is included on page 4-10) will cause their value to be added to the INCREMENT.

One of a restricted set of six character symbols appearing as an element in a MUST subfield will cause one of a corresponding set of MOBILE Data Reference indicators to be placed in the SYMBOL. This set of symbols is given in Section 4. 2. 4 of MUST Subroutine Symbols.

One of a restricted set of six character symbols appearing as an element in a MUST subfield will cause one of a corresponding set of CAP Subroutine Reference indicators to be placed in the SYMBOL This set of symbols is given in Section 4. 2. 4 of MUST Subroutine Symbols.

The appearance in a MUST subfield of a location symbol as the first element will cause a MOBILE Data Self-Reference indicator bit to be placed in the SYMBOL. The value of that symbol's location relative to the start of the unit of object code will be added to the INCREMENT. The appearance in a MUST subfield of a location symbol as any element but the first element will merely cause its value to be added to (or subtracted from) the INCREMENT.

The appearance of a double asterisk in a MUST subfield will cause a special indicator to be placed in the SYMBOL. (The appearance of an equal sign in a subfield will cause the remainder of the subfield to be completely ignored and will have the same effect as a double asterisk.)

On return from the subroutine, which thus evaluates a subfield, MUST will use the SYMBOL and the INCREMENT to construct the pertinent subfield of the CAP. The manner in which this is done will now be described for all MUST symbolic operation codes.

4.2.3 The Construction of a CAP Subfield by MUST

All of the MOBIDIC machine operation codes are obtainable through CAP language and therefore may be obtained via MUST by using the symbolic operation codes that are acceptable to DUST. Note that "extended" operation codes and "simulated machine instruction" codes (such as LDQ, CSZ) are not accepted. The majority of MUST operation codes may be followed by an "a, g, b" type of variable field. In such a case, a double asterisk indicator found in SYMBOL will cause the corresponding part number to be set in the CAP instruction and the INCREMENT to be ignored. A self-reference indicator found in SYMBOL will cause the current location counter to be subtracted from the value found in INCREMENT and the result stored as an increment or decrement to a self-reference operand. A MOBILE Data Reference Indicator found in SYMBOL will cause the INCREMENT to be stored as an increment or decrement to the MOBILE Data Reference Operand. A Subroutine Reference Indicator will cause the INCREMENT to be stored as an increment or decrement to the Subroutine Reference Operand. Note that with g and b subfields, the SYMBOL is ignored if it is not a double asterisk indicator and the INCREMENT is stored directly into the CAP instruction.

If a MOBILE Data Reference Operand is found in SYMBOL, the value in INCREMENT is added to the increment already found in the MOBILE Data Reference Operand. Thus, it is possible to define the nth location following a MOBILE Data Symbol by a unique name: the table symbol plus an increment (e.g., FILDAT + 12) may be used interchangeably with a name having some mnemonic meaning (e.g., AUXBFF). Either use will yield a CAP instruction with the MOBILE data symbol for FILDAT and an increment of twelve. A further increment to AUXBFF may also be used (e.g., AUXBFF + 1), but does not seem necessary.

The list of MUST operation codes and the corresponding CAP operation codes for which an a, g, b variable field is expected is given below:

ADB/024	ADD/012	ADM/013	CAM/011
CLA/010	CLS/014	CSM/015	CYL/035
CYS/034	DVD/022	DVL/023	HLT/000
LGA/003	LGM/002	LGN/004	LOD/051
LDX/053	MLR/021	MLY/020	MSK/055

NRM/037	RPA/054	RPT/001	SBB/025
SBM/017	SEN/005	SHL/030	SHR/032
SLL/031	SNR/007	SNS/006	SRL/033
STR/050	SUB/016	TRC/047	TRL/041
TRN/046	TRP/044	TRS/042	TRU/040
TRX/043	TRZ/045	TRA/140	

Note that if a b subfield occurs in a LOD instruction, it will have an octal 7740 logically added to it before being placed in the CAP instruction's b field.

Four other MUST operation codes are accepted which may have an a, g, b variable field. The g-b subfield is treated like the a field. Note that in this case, if a Subroutine Reference is required in either subfield, an increment to it cannot be given since the CAP format does not allow it. The operation codes are as follows:

MOV/052	PZE/105	MZE/106	MVA/152

In addition, it should be noted that a Subroutine Reference in the a field may not have an increment due to the limitations of the CAP format.

Three additional MUST operation codes are accepted and give rise to the corresponding CAP operation codes. They are explained below:

OCT/103	BES/101	BSS/102

The variable field of the OCT code may consist of the double asterisk or 0-12 octal characters preceded by a sign. Only one subfield is accepted. The number will be right justified after conversion to binary.

The variable field of BSS and BES codes may consist of a double asterisk or a number. Only one subfield is accepted. The number will be stored in the CAP instruction. Note that if the double asterisk is used, any instructions that refer to instructions on the "other side" of the BSS or BES CAP instruction via self-reference operands will probably not contain the correct increments.

Five additional MUST operation codes are accepted which do not give rise to CAP instructions but are actually instructions to the MUST translator. They are ORG, REM, SYN, END, and FIN.

The ORG code signifies that a new packet of coding is to begin. Three subfields must appear in the variable field, L, m, and n. The subfield "L" signifies the location in the CAP Skeleton Table at which the packet of coding will begin. If L is an asterisk, the first CAP instruction of the packet of coding will start after the last CAP instruction of the previous packet of coding (or the space reserved by its END card). The L subfield allows a packet of coding to be translated and inserted into an existing CAP Skeleton Table. The second subfield of an ORG variable field is a numeric field that gives the macro modification type. This field must be zero for subroutines. The third subfield, n, gives the macro or subroutine name in symbolic notation listed in paragraph 4. 2. 4.

The REM code is ignored by MUST and is merely passed onto the listing.

The SYN code allows the user of MUST to place a location type of symbol (SB2) into the symbol table. The variable field may consist of any predefined expression. The low order 15 bits of the INCREMENT will be placed into the symbol table as the value of the symbol, and SYMBOL will be ignored.

The END code signifies that the end of a packet of object coding (macro or subroutine) has been reached. The CAP instructions for the CAP Skeleton Table are punched and the correct directory word made up. If a number appears in the variable field END, it will be interpreted as the number of CAP instructions for which room is to be left in the CAP Skeleton Table. Thus, during debugging, the CAP Skeleton Table may be left in an expanded form. Corrections may be made via binary corrections or the entire packet of MUST coding may be retranslated and inserted into the table. If SFF16 is set during MUST-to-CAP translation, the variable field will be ignored and a compressed CAP Skeleton Table will be made up during translation of END.

The FIN code signifies that all the packets of coding have been translated.

4.2.4 MUST Subroutine Symbols

The reserved MOBILE data symbols and their CAP MOBILE Data Reference are as follows:

	CAP		
MUST Symbol	Key	Symbol	Increment
AUXBFF	2	2100	14
BUFFSZ	2	2100	17
BUFOVF	2	2100	27
CMPBSZ	2	2100	25
CURRIX	2	2100	15
CYCLWD	2	2100	21
DATASZ	2	2100	16
FILDAT	2	2100	00
MFRDAT	2	2200	00
RDRCIN	2	2100	26
RDRCSZ	2	2100	23
RELSWD	2	2100	22
RORDER	2	2100	20
TALLYS	2	3200	00
TAPDAT	2	2300	00
WORDER	2	2100	20
WTRCIN	2	2100	27
WTRCSZ	2	2100	23

The reserved subroutine symbols and their CAP Subroutine Reference operands are as follows:

Subroutine Symbol	English Subroutine Name	MUST Symbol
01000	Input-output Executor Routine	OTRAP
01001	Compute Type-2 Address Function	SKELA
02000	First I/O Constant Pool	AUXSB
02001	Compute Type-4 Address Function	SKELB
03000	Second I/O Constant Pool	AUXSC
03001	Compute Type-5 Address Function	SKELC
04000	First Temporary Storage Pool	AUXSD
05000	Second Temporary Storage Pool	AUXSE
06000	Open Routine	AUXSF
07000	Close Routine	AUXSG

Subroutine Symbol	English Subroutine Name	MUST Symbol
10000	Read-Write Fixed File	AUXSH
11000	Change to Next Reel	AUXSI
14000	I/O Save Subroutine	AUXSL
15000	Write Type Variable	AUXSM
16000	Handle Single Buffer	AUXSN
17000	Handle Double Buffers	AUXSP
20000	I/O Algorithmic Formula	AUXSQ
21000	Prepare to Write	AUXSR
22000	Write Last Block	AUXSS
23000	End-of-File Routine	AUXST
24000	Continuation Routine	AUXSU
25000	Check-a-Tape Routine	AUXSV
26000	Save and Restore Flip-Flops	AUXSW
27000	Write Header or Trailer Block	AUXSX
30000	I/O Binary-to-FIELDATA	AUXSY
31000	Initialize Standard Printout	AUXSZ
32000	Compute Checksum	AUXSA
33000	Check for Rerun	BUFAD
66000	Bulk Move Subroutine	BLKMV
67000	Editing Subroutine	EDITS
70000	Zero Divisor Test	ZDTST
71000	FIELDATA Rounded	FDRND
72000	Isolation B Routine	CHRIS
73000	General Storage	GNSTR
74000	Isolation Subroutine	ISOLT
75000	Examine Character Subroutine	EXMNE
76000	Binary-to-FIELDATA Conversion	BINFD
77000	FIELDATA-to-Binary Conversion	FDBIN

Note that these MUST subroutine symbols are only 5 characters in length. A sixth character must be appended when the symbol is used, and will be treated as an octal character and stored in the CAP subroutine symbol in the "internal subroutine symbol" bits. This table will be expanded to incorporate the symbols of more subroutines or to take account of duplicate MUST symbols for the same CAP subroutine.

The list of standard symbols and their octal equivalents is as follows:

Symbol	Octal
IR1	77741
IR2	77742
IR3	77743
IR4	77744
ACC	77750
QRG	77751
BRG	77752
PCT	77753
PCS	77754
WSR	77760
ZERO	77740
SFF1-SFF16	110-127

4.3 RUN 4 OUTPUT TABLE FORMATS

This section contains the formats of all the Data Replies, Data Analyzer Table, Address Function Table, and Address Variable Table.

4.3.1 Data Replies

When it has been determined what type of data reply is to be made to any query within a compressed generator call, and when this reply has been appended to the call, a 3-bit key is necessary to inform Run 3 of the reply type.

Type of Reply	Key	
Standard reply	000	ARITHMETICS
Bulk reply	001	MOVE
Matching reply	010	MOVE CORRESPONDING
Procedure reply	011	GO TO
Hardware reply	100	IF STATUS
Condition reply	101	IF RELATION

4.3.1.1 Standard Data Reply

1. Table name—Standard Data Reply.
2. Table symbol—none, an element of GB4N.
3. Number of words/entry—variable, dependent on editing information of data-unit queried.
4. Number of entries—dependent on number of queries on fields.
5. Table function—used by the generators for arithmetic statements.
6. Table format—

Word 1:

Bits 1-3	Zero
Bits 4-9	File number
Bits 10-21	Fixed increment

6. Table format—(Cont.)

Bits	22-23	Not used
	24	Specifies negative fixed increment (yes = 1, no = 0)
	25-26	Not used
	27-36	Final address function serial number

<u>Word 2:</u>

Bits	1-3	Standard reply key (000)
	4	Editing information present (no = 0, yes = 1)
	5	Picture present (no = 0, yes = 1)
	6	"Justified" clause given (no = 0, yes = 1)
	7	Does data obey standard rules of justification? (yes = 0, no = 1)
	8	Is data unit synchronized? (no = 0, yes = 1)
	9	Direction of synchronization (right = 0, left = 1)
	10	If Picture is present, can all information given in Picture be represented by bits? (yes = 0, no = 1)
	11	Assumed point location: direction to move to find point location (left = 0, right = 1)
	12-18	Assumed point location: number of places from low order end to find point
	19	Is there a sign-name? (no = 0, yes = 1)
	20-24	Not used
	25-26	Base (binary = 01, FIELDATA = 10, mixed = 11)

Bits	27	Is class given? (no = 0, yes = 1)
	28	If class is given, type (numeric = 0, non-numeric = 1)
	29	Not used
	30-31	Recording mode (NISN = 00, ISN = 01, either = 10 or 11)
	32	Is there a "signed" clause? (no = 0, yes = 1)
	33-36	Type of sign (See Word 3 (33-36) in Section 4. 4. 2)

Word 3:

Bits	1	Is size of data unit constant? (yes = 0, no = 1)
	2-3	Not used
	4-5	Units of size (bits = 00, FIELDATA characters = 01, words = 10)
	6-20	Maximum size of data unit—FIELDATA base
	21	Starting bit information (fixed = 0, varying = 1)
	22-27	Net starting bit—FIELDATA base
	28-36	Jump to reply on "Sign Name" (zero if no "Sign Name")

Word 4:	Format is the same as Word A of DDT. See paragraph 4. 4. 2. NOTE: This word is present if Bit 4 of Word 2 above =1.
Word 5—Word N:	Format is the same as Word D of DDT. See Paragraph 4. 4. 2 NOTE: These words are present if Bit 5 of Word 2 above =1.
Word N+1—Word K:	Format is the same as above since a sign name is a data-name; therefore a standard data reply is made. NOTE: These words are present if Bit 19 of Word 2 = 1.

4.3.1.2 Bulk Reply

1. Table name–Bulk Reply.
2. Table symbol–none, an element of BG4N.
3. Number of words/entry–variable, dependent on size information of data unit queried.
4. Number of entries–dependent on number of queries on non-elementary items in "MOVE" calls.
5. Table function–used by the generators for MOVE statements.
6. Table format–

Word 1:

Bits	1-3	Zero
	4-9	File number
	10-21	Fixed increment
	22-23	Not used
	24	Specifies negative fixed increment (yes = 1, no = 0)
	25-26	Not used
	27-36	Final address function serial number

Word 2:

Bits	1-3	Bulk reply key (001)
	4-5	Unit of size (bits = 00 FIELDATA character = 01, words = 10)
	6-20	Maximum size of data-unit
	21	Starting bit information (fixed = 0, variable = 1)
	22-27	Net starting bit
	28	Does data contain variable blocks? (no = 0, yes = 1)
	29-30	Base (binary = 01, FIELDATA = 10, mixed = 11)

Bits	31	Has class been given? (no = 0, yes = 1)
	32	If class is given, type (numeric = 0, non-numeric = 1)
	33-36	Not used

If there are variable blocks as specified in Word 2, Bit 28, Word 2 is followed by one word or a variable number of words for each fixed block or variable block respectively in their given order.

Fixed Block Format:

Bits	1	Specifies a fixed block (0)
	2	Not used
	3	Is this the last block? (no = 0, yes = 1)
	4-21	Not used
	22-36	Size of block, in bits.

Variable Block Format:

Word 1:

Bits	1	Specifies a variable block (1)
	2	Type of control (count control = 0, sentinel control = 1)
	3	Might block not be present? (no = 0, yes = 1)
	4	Is this the last block? (no = 0, yes = 1)
	5-6	Not used
	7-21	Maximum number of occurrences
	22-36	Fixed size of repeated units, in bits

The following two words occur next if the variable block is under count control:

Word 2:

Bits	1	Base of count field (binary = 0, FIELDATA = 1)

Bits 2	Starting bit information (fixed = 0, variable = 1)
3-18	Not used
19-24	Net starting bit
25-36	Size of count field in characters or bits

Word 3:

Bits 1-3	Not used
4-9	File number
10-21	Fixed increment
22-23	Not used
24	Specifies negative increment (yes = 1, no = 0)
25-26	Not used
27-36	Final address function serial number

The following words occur if the variable block is under sentinel control:

<u>Word 2:</u>

Bits 1-6	Starting word in repeated unit of sentinel word
7-12	Starting bit
13-21	Size of sentinel field, in bits
22-26	Not used
27-36	Number of Value words following
<u>Word 3—Word N:</u>	Value words—Format is the same as Word E-N of DDT. See paragraph 4.4.2.

4.3.1.3 Matching Data Reply

1. Table name—<u>Matching Data Reply</u>
2. Table symbol—none, an element of GB4N
3. Number of words/entry—variable, depends on hierarchy and size of the names.
4. Number of entries—dependent on number of queries on data units within a <u>Move Corresponding</u> verb.
5. Table function—used by the generators for MOVE CORRESPONDING statements.
6. Table format—

<u>Word 1:</u>

Bits 1-3	Matching reply key (010)
4-12	Parent jump, relative to first generator call header word
13	Specifies if data unit has descendant (yes = 0, no = 1)
14	Elementary item (yes = 1, no = 0)
15-16	Not used
17-19	Number of words following for its data-name

Bits	20-21	DDT memory load number
	22-36	DDT entry address

<u>Word 2–Word N</u>:

Bits	1-6	First character of data-name
	7-12	Second character of data-name
	.	
	.	
	.	

etc., as many words as necessary to express this data-name with binary-zero fill to the end of the word.

NOTE: Each descendant in the hierarchy to the data-name queried will have the same format and will succeed that data-name.

4.3.1.4 Procedure Data Reply

1. Table name–<u>Procedure Data Reply</u>
2. Table symbol– none, an element of BG4N
3. Number of words/entry– 1
4. Number of entries–dependent on number of queries on procedure names.
5. Table function–used by the generators for PERFORM, GO TO, ALTER statements.
6. Table format–

Bits	1-3	Procedure reply key (011)
	4-6	Not used
	7-21	Procedure beginning FISN
	22-36	Procedure ending FISN

4.3.1.5 Hardware Data Reply

1. Table name–<u>Hardware Data Reply</u>
2. Table symbol–none, an element of GB4N

3. Number of words/entry—1
4. Number of entries—dependent on number of queries on hardware names.
5. Table function—used by the generators for IF STATUS statements.
6. Table format—

Bits	1-3	Hardware reply key (100)
	4	On-Off Status (on = 0, off = 1)
	5-6	Not used
	7-12	Hardware number (low order 6 bits of MOBIDIC device designation)
	13-36	Not used

4.3.1.6 Condition Reply

1. Table name—Condition Reply
2. Table symbol—none, an element of GB4N
3. Number of words/entry—variable, dependent on editing information of data unit queried
4. Number of entries—dependent on number of queries on condition names
5. Table function—used by the generators for IF RELATION statements
6. Table format—

Word 1:

Bits	1-3	Condition reply key (101)
	4-9	Number of values (literals)
	10-24	Number of bits to express each value
	25-36	Not used

<u>Word 2—Word N:</u>		Value words—Format is the same as Word E-N of DDT. (See paragraph 4.4.2.)
<u>Word N + 1:</u>		
Bits	1-3	Not used
	4-9	File number
	10-21	Fixed increment
	22-23	Not used
	24	Specifies negative fixed increment (yes = 1, no = 0)
	25-26	Not used
	27-36	Final address function serial number
<u>Word N + 2:</u>		Same as Word 2 of Standard reply
<u>Word N + 3:</u>		Same as Word 3 of Standard reply
<u>Word N + 4:</u>		Same as Word 4 of Standard reply
<u>Word N + 5—Word R:</u>		Same as Word 5—Word N Standard reply
<u>Word R + 1—Word L:</u>		Same as Word N + 1—Word K Standard reply

4.3.2 <u>Data Analyzer Table</u>

1. Table name—<u>Data Analyzer Table</u>
2. Table symbol—TK45 produced by Run 4
3. Number of words/entry—3
4. Number of entries—dependent on number of data replies made by Run 4.
5. Table function—used by Run 8 to produce the Data Analyzer Listing
6. Table format—

<u>Words 1 and 2:</u>	Format is the same as Words 1 and 2 of the final TK45. (See paragraph 4.5.9.)

Word 3:

Bits	1-36	Not used

4.3.3 Address Function Table

1. Table name—Address Function Table
2. Table symbol—TA3N produced by Run 4
3. Number of words/entry—3
4. Number of entries—dependent upon the complexity of the Data Division
5. Table function—used by Run 6 to produce "computes" and "loads of index registers" at object time.
6. Table format—

Word 1:

Bits	1-6	Address function key (type 2 = 2, type 4 = 4, type 5 = 6)
	7-21	Key size: octal 3: words in length (3)
	22-36	Item size: octal 3: words per item (3)

Word 2:

Bits	1-3	Function type (type 2 = 010, type 4 = 100, type 5 = 110)
	4-20	Type 2: not used Type 4: value in words Type 5: value in bits
	21	Not used
	22-36	Type 2: not used Type 4 or Type 5: Bits 22 0 = working storage; 1 = special storage 23-36: Fixed word increment within storage area

Word 3:

Bits	1-6	File number

7-16	Type 4: not used Type 2: number of jump word Type 5: number of jump word Bits 7 : 0 = NOT first type 5 address function 1 = IS first type 5 address function 8-10: not used 11-16: starting bit of address function
17-26	Previous address function number
27-36	Serial number of this address function Type 2: used for a block address after a variable block Type 4: used with an Occurs given in words Type 5: used with an Occurs given in bits

4.3.4 Address Variable Table

1. Table name—Address Variable Table

2. Table symbol–produced by Run 4 and left in core.

3. Number of words/entry—1

4. Number of entries—dependent upon the number of subscripts used in the Procedure Division Source program

5. Table function—used by the generators for optimization

6. Table format—

Bits 1-20	Not used
21	Integer literal (yes = 1, no = 0)
22	Working storage = 0 Special storage = 1
23-36	Fixed increment within this storage area

4.4 DATA DESIGN TABLE

This section contains the Data Design Table format used by RUN 4.

4.4.1 DDT Format for File Description

Word 0:

Bits 1-3	DDT Entry ID File: = 010
4-15	Maximum block size 12 Note: Minimum block size is found in Word 3.
16-21	Label "VALUE" Jump Address 6-bit address – relative to DDT memory load.
22-36	Location of next DDT entry 15-bit address – relative to DDT memory load.

Word 1:

Bits S	Is the block variable or fixed? (0 = fixed, 1 = variable)
1	Not used
2	Are there labels? (0 = yes, 1 = no)
3	Is there more than one type of data record? (0 = yes, 1 = no)
4	Is the complete file description in this memory load? (0 = yes, 1 = no)
5-6	Units of block size (00 = FIELDATA characters, 01 = records, 10 = words)
7-8	Address currency information (01 = yes, 10 = no)
9	Was this file description taken from the library? (0 = no, 1 = yes)
10-11	Recording Mode (00 = NISN, 01 = ISN)
12	Not used
13-18	Location within this entry of the file's name
19-24	Location within this entry of the renamed file-name. If 0 implies that there is no renamed file-name.
25-30	Not used
31-36	File number

Word 2:

Bits S	Is record size fixed? (0 = yes, 1 = no)
1	Not used
2	Units of record size (0 = characters, 1 = words)
3	Is a header present? (0 = no, 1 = yes)
4-5	Is the trailer or hash-total present? (00 = neither, 11 = trailer is present, 10 = hash-total is present)
6	Not used
7-21	Maximum record size
22-36	Minimum record size

Word 3:

Bits S	Not used
1-12	Minimum block size
13-36	File size in number of records if given

Word 4-? Descendants

Bits S	1 = last word; 0 = more words follow
1	Is the record, located by bits 7-21, a label record? (0 = no, 1 = yes)
2	Is the record, located by bits 22-36, a label record? (0 = no, 1 = yes)
3-6	Not used
7-21	Location in this memory load of the DDTentry for a record contained in this file
22-36	Location in this memory load of the DDT entry for a record contained in this file

Note: This field might be zero in the last word if there is an odd number of record types in the file.

Word A – ? File-name (1 to 5 words)

Located by bits 13-18 in Word 7. The first character of the file-name is in bits 1-6 of Word A; the second, in bits 7-12 of Word A; etc. The sign bit (1) indicates last word with master-space-fill on the right.

Word B – ? Renamed File-name (1 to 5 words)

Located by bits 19-24 in Word 1. The first character of the NAMED file-name is in bits 1-6 of Word B; the second, in bits 7-12 of Word B; etc. The sign bit indicates last word. The Renamed File-name may not be present.

Word C – X Value Information

If the data is present, it is located via bits 16-21 in Word O. The first character of the Value is located in bits 1-6 of Word C; the second character is located in bits 7-12 of Word C; etc. Sign bit (1) indicates last word.

4.4.2 Record, Field DDT Entry Formats

Word 0:

Bits 1-3	DDT entry ID. For record field: 000
4	Is this a field? (0 = yes, 1 = no)
5	Is this data-unit redefined? (0 = no, 1 = yes) Note: Illegal for levels 01, 66
6	Is this data-unit in a label record? (0 = no, 1 = yes)
7-12	Location of editing information
13-21	Size of previous DDT entry
22-36	Location of next DDT entry

Word 1:

Bits S	Is there any editing information? (0 = no, 1 = yes)
1-6	Adjusted level-number Note: Level-number 77 will be octal 77 Level-number 66 will be octal 66

7-12	File-number	
13	Is data-unit a simple subscript? (0 = no, 1 = yes)	
14	Does this label field have a value? (0 = no, 1 = yes)	
15	Is this data-unit in a labeled-item record? (0 = no, 1 = yes)	
16-18	If this data-unit is in a label, what type of label is it? Values: 000 = BEGINNING FILE LABEL 001 = BEGINNING TAPE LABEL 010 = END OF FILE LABEL 011 = END OF TAPE LABEL 100 = OTHER	
19	Is this field used to find the number of occurrences? (0 = no, 1 = yes)	
20	Is this field used to find the size? (0 = no, 1 = yes)	
21	Is this field a "sign-name?" (0 = no, 1 = yes)	Used by Run 1.2
22-36	Location of this data-unit's parent's DDT entry	

Word 2 (contains jumps to variable information)

Bits S	Elementary item (1 = yes, 0 = no)
1-6	Location of this data-unit's data-name (Word B)
7-12*	Location of this data-unit's descendants (Word C)
13-18*	Location of this data-unit's Picture (Word D)
19-24*	Location of this data-unit's Value (Word E) Note: A Value, given by Source Program for a data-unit in the file section, will be ignored.
25-30*	Location of either: A. The Data-name Address Word (DNAW) for the data-name in the "OCCURS DEPENDING UPON" option (Word F) or B. The Size information, given at the record level only (Word G)
31-36*	Location of "JUMP–information word" giving jumps to A. The Range (Word I) B. Sign-name (Word J)

*If the 6 bits are zero, then the item is not present.

Word 3 Arithmetic Information

Bits 1	Has the Justified clause been given? (0 = no, 1 = yes)
2	Does the data obey the standard rules of Justification? (0 = yes, 1 = no) Note: The standard rules are: Numeric data: right justified, zero fill on left Non-numeric data: left justified, space fill on right
3	Is data-unit synchronized? (0 = no, 1 = yes)
4	Left or right synchronization? (0 = right, 1 = left)
5	If PICTURE is present, then can all information given in the picture be represented by bits? (0 = yes, 1 = no)
6	Assumed Point Location: Direction to move to find point location (0 = left, 1 = right)
7-13	Assumed Point Location: Number of places from low order end to find point
14	Is there a sign-name? (0 = no, 1 = yes)
15-18	Not used
19-24	Not used
25-26	Data Base (01 = binary, 10 = FIELDATA, 11 = Mixed)
27	Has the Class been given? (0 = no, 1 = yes)
28	If class has been given (0 = numeric, 1 = non-numeric)
29	Not used
30	Recording Mode given (0 = no, 1 = yes)
31	Recording Mode (if bit 30 = 1) 0 = NISN, 1 = ISN
32	Is there a SIGNED clause? (0 = no, 1 = yes)
33-36	Type of Sign 0 = No sign indicated 1 = Sign bit (bit 32 must be 1) 2 = High order "+" (if bit 32 = 1, implies high order sign). 3 = High order "-" 4 = CR 5 = DB 6 = Floating "+" 7 = Floating "-"

Note: See PICTURE clause in section 5.2.9 of the COBOL Preliminary Reference Manual for meanings of "+ ", "-", CR, DB

Word 4 Size and Occurrence Information

Bits S	Is the Size of this data-unit constant? (0 = yes, 1 = no)
1-2	Units of Size (00 = bits, 01 = FIELDATA characters, 10 = words)
3-6	Not used
7-21	Maximum Size of data-unit Note: Size will include the operational sign if not in sign bit
22-36	Maximum number of occurrences

Word 5 Occurs and Addressing Information

Bits S	Is there any number of occurrences? (0 = no, 1 = yes)
1-6	If input: 2nd step, number of bits (or starting bit) (i.e., bit 1, 2, 3, . . .)
7-12	If input: 1st step, in words (to get around A words) (i.e., word 0, 1, 2. . .)
13	Is there a variable number of occurrences? (0 = no, 1 = yes)
14	If there is a variable number of occurrences, how can the exact number be found? (0 = count field, 1 = sentinel field)
15	Are there any A Words present? (0 = no, 1 = yes)
16	Is the addressing information record or parent relative? (0 = record relative, 1 = parent relative)
17	If input: Leap? or (exactly equivalent) Is the addressing information for the input not the same as for output? (0 = no, 1 = yes)
18-21	Not used
22-36	If input: 2nd step, number of words (or starting word) (i.e., word 0, 1, 2, . . .)

Notes on Word 5

1. If an input record is parent relative (bit 16 = 1), then the 1st step (bits 7-12) and Leap bit (20) have no meaning.

2. If the addressing information for the input is the same as the output (bit 17 = 0), then

   1st step (bits 7-12) = number of A Words present in record (to jump around)

   2nd step (bits 1-6, 22-36) = distance from 1st word following A Words (if any) to 1st bit of data-unit

3. The method of obtaining the correct addressing information is described below:

<u>If Input Record</u>

If Parent Relative (bit 16 = 1): Starting word, bit (bits 22-35, 1-6)

If Record Relative (bit 16 = 0):

1. If A Words are not present (bit 15 = 0):
   then, starting word in bits (22-36) starting bit in bits (1-6)

2. If A Words are present (bits 15 = 0): If no Leap (bit 17 = 0):
   then, starting word = first step (bits 7-12) + second step (bits 22-36) starting bit in bits (1-6)

   If Leap (bit 17 = 1): (as above)

<u>If Output Record</u>

If input ≠ output (bit 17 = 1), starting word and bit can be found in Word 7, bits 22-36, 1-6.

If input = output (bit 17 = 0), starting word and bit can be found in Word 6, bits 22-36, 1-6.

Note: Check bit 16 for record or parent relative.

<u>Word 6 Size, Occurs, Addressing Information</u>

Word 6 is present only if there are any variable occurrences (Word 5, bit 13 = 1), variable size (Word 4, sign bit = 1), or input not equal output addressing information (Word 6, bit 17 = 1).

Bits 1-6	If output: 2nd step, number of bits (or starting bit)
7-21	Minimum size Note: Same units as maximum size in Word 5.
22-36	If output: 2nd step, number of words (or starting word)

Word 6 or 7 or 8 Label Record Information

Word 6, 7, or 8 may not be present. Data-unit must be in a label record (Word 0, bit 6 = 1) and must have a value (Word 1, bit 14 = 1).

Is Word 6 if: data-unit is constant in size (Word 4, sign bit = 0) and if it has no variable occurrences (Word 5, bit 13 = 0)

Is Word 7 if: data-unit is variable in size (Word 4, sign bit = 1) and if it has no variable occurrences (Word 5, bit 13 = 0)

Is Word 8 if: data-unit has a variable number of occurrences (Word 5, bit 13 = 1)

Bits	1	Is there a literal or "data-name-4?" (0 = literal, 1 = data-name 4)
	2	Is there a hashed option? (0 = no, 1 = yes)
	3-21	Not used
	22-36	Location within this memory load of data-name-4 or if it is a literal, location within this entry of the literal (bits 31-36 only).

Word 7 Occurrence Information

Word 7 may not be present. Word 7 is needed only when there is a variable number of occurrences (Word 5, bit 13 = 1). Word 6 will be present.

Bits	1-21	Not used
	22-36	Minimum number of occurrences

Word A Editing Information

Word A might be present. Presence of Word A is indicated by Word 1, sign = 1. Word A is located by bits 7-12 in Word O.

Bits	1	Is there zero suppression? (0 = no, 1 = yes)
	2	Is there check protection? (0 = no, 1 = yes)
	3	Is there a floating dollar sign? (0 = no, 1 = yes)
	4	Is there a high order dollar sign? (0 = no, 1 = yes)
	5	Is there a "blank when zero option?" (0 = no, 1 = yes)

Bits 6	Not used
7-12	If there is zero suppression, check protection, or a floating dollar sign, how many characters preceding the decimal point will not be replaced by blanks if zero?
13	Are there standard commas? (0 = no, 1 = yes)
14	Is there a real decimal point? (0 = no, 1 = yes)
15-18	Not used
19-24	Number of places to move from low order end of field to find the real decimal point.
25	Is there a carriage return given? (0 = no, 1 = yes)
26	Is the carriage return leading or trailing? (0 = leading, 1 = trailing)
27	Is there a tab? (0 = no, 1 = yes)
28	Is the tab leading or trailing? (0 = leading, 1 = trailing)
29	Are there any control characters in the picture that cannot be located by bits 25-28? (0 = no, 1 = yes)
30	Are there any insertion characters which cannot be located by the above bits (such as zeros, blanks, non-standard commas, decimal points, and dollar signs)? (0 = no, 1 = yes)
31-36	Number of control characters

Word B up to Word B + 4 Data Unit's Data-Name

Note: Word B is located by bits 1-6 of Word 2. The last word is negative and space filled. First character of data-name will occupy bits 1-6 of 1st word; second, bits 7-12 of 1st; etc.

Word C-? List of Descendants for these Data-Units

Word C might be present. Word C is found by using bits 7-12 of Word 2. There will be one word for each set of descendants.

Bits S	If equal to 1, then last word
1-6	Not used
7-21	Location of the DDT entry for one of these data-units descendant
22-36	Location of the DDT entry for one of these data-units descendants

Note: If bits S-1 and bits 22-36=0, an odd number of descendants is indicated.

<u>Word D – ? The Picture for this Data-Unit</u>

Word D might be present. Word D is found by using bits 13-18 of Word 2.

Note: The first character of the Picture will occupy bits 1-6 of the first word; the second, bits 7-12 of the first, etc. The last word is negative and space filled.

<u>Word E – ? The value for this Data-Unit</u>

Word E might be present. Word E is found by using bits 19-24 of Word 2. The format is the same as Word D above.

<u>Word F The Data-Name Address Word for the Data-Name in the Occur's Depending Upon Option</u>

Word F might be present. Word F is found by using bits 25-30 of Word 2. Word F is present only if word 5, bit 13 = 1.

Bits 1-21	Not used
22-36	Location within this memory load of the DDT entry for the data-name in the Occur's Depending Upon Option

<u>Word G – ? Size Information for the Record</u>

Word G is present only for data-units which have level-number 01 and are in the file section. Word G is found by using bits 25-30 of Word 2.

Note: The only difference between the Size Information in the File Description Block and in the DDT is in bits 22-36 of Word 1, (for a variable block with sentinel control). In the DDT, the location of the sentinel value is relative to the beginning of the memory load.

<u>Word H A 2nd "Jump-information Word"</u>

Word H may be present. Word H is found by using bits 31-36 of Word 2. Word H is present only if there is a sign-name or range for this data-name.

Bits 1-6	Location of this data-unit's range*
7-12	Location of the DNAW for the sign-name*
13-36	Not used

* If the 6 bits are zero, then the item is not present.

Word I – ? The Range

Word I may be present. Word I is found by using bits 1-6 of Word H.

Note: The format of the Range is the same as for the Value except there are two literals. The first is the lower bound, the second, the upper bound. See Note for Word E above.

Word J The Data-Name-Address Word for the data-name in the "SIGN IS" Option

Word J may be present. Word J is found by using bits 7-12 of Word H.

Bits 1-21	Not used
22-36	Location within this memory load of the DDT entry for the Sign-name

4.4.3 DDT Formats for Condition-Name

Word 0:

Bits 1-3	DDT entry ID for condition-name: 001
4-6	Not used
7-12	Location within this entry of the condition-name
13-21	Size of previous DDT entry
22-36	Location of the next DDT entry

Word 1:

Bits 1-6	Number of literals
7-12	The number of bits to express each literal
22-36	Location in DDT of Parent

Word A up to Word A + 4 The Condition-name (1 to 5 words)

Note: Each condition-name may have more than 1 literal associated with it. The literals will be in ascending order. A range is considered as two literals.

Each literal will be right synchronized. In the first word of each literal, the first 6 bits will contain information about the the literal. If bit 1 = 1, this means that this literal is the lower bound of a range. The next literal will then be the upper bound.

If the conditional variable has a high order sign, then the first character of the literal will be the sign. If it has a sign bit or "sign-name, " then the literal's sign is indicated by bit 6 in the first word. The number of machine words that a literal occupies is as follows:

Literal size in bits	Machine words
1-30	1
31-66	2
67-102	3
.	.
.	.
.	.
2115-2160	61

One to 360 FIELDATA chpracters are permissable for each literal.

4.4.4 DDT Formats for Procedure-Name

Word 0:

Bits 1-3	DDT entry ID for procedure-name: 100
4	Is this a paragraph or section-name? (0 = paragraph, 1 = section)
5	Is this procedure-name duplicated? (0 = no, 1 = yes)
6	Not used
7-21	First "short FISN" in procedure
22-36	Location of next DDT entry

Word 1:

Bits 1-6	Not used
7-21	Last "short FISN" in procedure
22-36	Location of section-name in DDT
Note:	This has no meaning if bit 4 of Word zero is 0.

Words 2 up to 6 Procedure-Name (1 to 5 words)

The last word of the procedure-name will be negative.

4.4.5 DDT Format for Condition-Names for Switches

Word 0:

Bits	1-3	DDT entry ID for switch condition-names: 011
	4	On-Off status (0 = on, 1 = off)
	5-6	Not used
	7-12	Hardware-number
	13-18	If bit 4 = 1, it is the size of the preceding DDT entry
	19-21	Not used
	22-36	Location of next DDT entry

Note: This is used to find other status DDT entry. If this entry is ON: See bits 22-36 for OFF status. If this entry is OFF: ON status can be obtained by subtracting number in bits 13-18 from this entry's origin.

Word 1 up to Word 5

The condition-name for this switch status. First character will occupy bits 1-6 of 1st word; second, bits 7-12 of 1st; etc. The final word will be negative. Final word will be left justified with right master space fill.

4.5 RUN 8 TABLE FORMATS

This section contains the format of every input table used by Run 8.

4.5.1 Block Source Program (BSP)

1. Table name – Block Source Program (BSP)
2. Table symbols – LPON set by RUN 1.0 and RUN 1.1
   LPON set by RUN 1.2
   LPON set by RUN 1.3
3. Number of words/entry – 14
4. Number of entries – dependent on Source Program
5. Table function – used by RUN 8 to produce a Block Source Program listing.
6. Table format –

Word 1.	6 FIELDATA characters columns	1-6 of Source Program
2.	6 FIELDATA characters columns	7-12 of Source Program
3.	6 FIELDATA characters columns	13-18 of Source Program
4.	6 FIELDATA characters columns	19-24 of Source Program
5.	6 FIELDATA characters columns	25-30 of Source Program
6.	6 FIELDATA characters columns	31-36 of Source Program
7.	6 FIELDATA characters columns	37-42 of Source Program
8.	6 FIELDATA characters columns	43-48 of Source Program
9.	6 FIELDATA characters columns	49-54 of Source Program
10.	6 FIELDATA characters columns	55-60 of Source Program
11.	6 FIELDATA characters columns	61-66 of Source Program
12.	6 FIELDATA characters columns	67-72 of Source Program
13.	6 FIELDATA characters columns	73-78 of Source Program

Word 14:

Bits 1-12	2 FIELDATA characters columns	79-80 of Source Program

13-24	binary card count
25-36	Identification code – FIELDATA BB

4.5.2 Expanded Source Program (ESP)

1. Table name – Expanded Source Program (ESP)
2. Table symbol – LP1N set by RUN 1.3
3. Number of words/entry – dependent on size of Procedure Division Source Program Sentence
4. Number of entries – dependent on Source Program
5. Table function – used by RUN 8 in producing an Object Program Assembly listing
6. Table format

Word 1-n:

6 FIELDATA characters of Procedure Division Source Program per word. Word n has blank fill (05).

Word n+1:

Bits	1-18	not used
	19-24	FIELDATA C or zero*
	25-36	Identification code – FIELDATA EE

4.5.3 BSP1 Correction Table

1. Table name – BSP1 Correction Table
2. Table symbols – TX08 set by RUN 1.0 + 1.1
   TX0N set by RUN 1.2
   TX1N set by RUN 1.3
3. Number of words/entry – 1

*FIELDATA C is used whenever a table entry is continued across tape blocks.

4. Number of entries – dependent on number of errors found by the translation and generation phases in the Source Program.

5. Table function – used by RUN 8 in producing a Block Source Program listing with errors.

6. Table format

Bits	1-12	not used
	13-24	binary card count
	25-36	binary error flag number

4.5.4 BSP2 Correction Table

1. Table name – BSP2 Correction Table

2. Table symbols – ER48 set by RUN 4
   ER38 set by RUN 3

3. Number of words/entry – 1

4. Number of entries – dependent on number of errors found by the generation phase.

5. Table function – used in an indirect way by RUN 8 with the FISN-card count table to produce a BSP1 Correction Table. This allows errors found in generation phase to appear in the BSP listing.

6. Table format

Bits	1-6	not used
	7-21	generator call FISN
	22-23	not used
	24-36	binary error flag number

4.5.5 FISN-Card Count Table

1. Table name – FISN-Card Count Table

2. Table symbol – CF18 set by RUN 1.3

3. Number of words/entry – 1

4. Number of entries – dependent on number of Procedure Division Source Program cards of input.

5. Table function – used by RUN 8 in major error handling. It links up cards containing Procedure Division Source Program statements with the generator calls produced.

6. Table format –

Bits	1-6	not used
	7-21	generator call FISN
	22-36	binary card count

4.5.6 Symbol Table

1. Table name – Symbol Table

2. Table symbol – TM7N set by RUN 7

3. Number of words/entry – 1

4. Number of entries in Part I-16

   Number of entries in Part II – dependent on the number of subroutines used by the generator.

5. Table function – used by RUN 8 in the CAP-to-binary conversion, Core Storage Allocation listing, Subroutine Assembly listing, and Object Program Assembly listing.

6. Table format for Part I –

Bits	1-6	area symbol
	7-21	binary size in words
	22-36	area starting address at object time

7. Table format for Part II –

Bits	1-6	not used
	7-21	subroutine starting address at object time
	22-27	subroutine name
	28-30	internal symbol
	31-36	duplication number

N.B. There are 16 areas of object code specified in Part I of the Symbol table:

1. Initialization
2. File Data Table
3. Input-output files
4. Working storage
5. Data Division constants
6. Multiple file reel data
7. Internal constants
8. Alter table
9. Address function values
10. Address function starting bits
11. Open storage
12. Special storage
13. Undefined exit
14. Subroutine instructions
15. Running program instructions
16. Running program END

In Part II of the Symbol Table the Subroutine name, internal symbol, and duplication number 0 or 1 comprise the Subroutine symbol. Internal symbol–ranges from 0-7.

If = 0, start of subroutine, otherwise an internal entry point of subroutine.

Duplication number–allows for same subroutine to be used repeatedly with minor revisions.

4.5.7 File Requirement Table

1. Table name–<u>File Requirement Table</u>
2. Table Symbol–TI97 set by RUN 7

3. Number of words/entry– 8

4. Number of entries–dependent on the number of file descriptions in the Data Division Source Program.

5. Table function–used by RUN 8 in the CAP-to-binary conversion, Core Storage Allocation listing, Subroutine Assembly listing, and Object Program Assembly listing.

6. Table format–

<u>Word 1</u>:

Bits 1-6	binary file number
7-12	number of words in this entry
13-14	I/O
15-16	CR
17	B
18	A
19	P
20-21	not used
22-36	file buffer area address at object time

<u>Word 2</u>:

Bits 1-5	size of current record area, currency index address, or zero
16-21	count of characters in file name
22-36	size of one file buffer

<u>Word 3</u>:

Bits 1-6	number of tape drives
7-36	truncated 5-bit tape drive numbers

<u>Words 4-8</u>:	file name characters with blank fill (05).

NOTES:

1. I/O = 01 if input file
   = 10 if output file
   = 11 if input-output file

2. CR = 00 if no current record area or currency index
 = 10 if current record area
 = 11 if currency index

3. B = 0 if single buffer
 = 1 if double buffer

4. Address in word 1 is initial buffer address.

5. CRA size – if CR = 10, then in RUN 8 the size of the current record area is contained here. RUN 8 replaces it by the starting address of the current record area.

6. CI address – if CR = 11, then RUN 8 places the address of the currency index here.

7. Buffer size – if B bit = 1, another buffer of same size immediately follows.

8. A = 0 unique area for this file
 = 1 this file shares same area in core as file in previous entry.

9. P = 1 if file is parallel processed (both files open at the same time) with file in preceding File Requirement Table entry.

4.5.8 Assigned Procedure Table

1. Table name – Assigned Procedure Table

2. Table symbol – TB7N set by RUN 7

3. Number of words/entry in Part I-2

   Number of words/entry in Part II – dependent on the number of characters in each paragraph or section name.

4. Number of entries – dependent on the number of paragraphs in Procedure Division Source Program.

5. Table function – used by RUN 8 in CAP-to-binary conversion, and Data Analyzer listing.

6. Table format:

   Part I

   Word 1:

Bits	1-6	Key
	7	name; 1 = section-name 0 = paragraph-name

8-15	not used
16-21	character count of procedure-name
22-36	jump address, relative to start of table, to procedure-name in Part I

Word 2:

Bits 1-15	new internal sequence no. (NISN)
16-21	not used
22-36	absolute starting address of this paragraph at object time

Part II

Words 1-n

6 characters of procedure name per word with master space fi'l (00).

N.B. ≤ 5 since procedure name must be ≤ 30 characters.

4.5.9 Data Analyzer Table

1. Table name - Data Analyzer Table
2. Table symbol - TK46 set by RUN 6
3. Number of words/entry - 3
4. Number of entries - dependent on the number of data units and procedure-names referred to in Procedure Division Source Program.
5. Table function - used by RUN 8 to produce the Data Analyzer listing.
6. Table format -

Data-Names

Word 1:

Bits	1-6	Data Analyzer sort key (00)
	7-21	key size = 3
	22-36	item size = 3

Word 2:

Bits	1	= 0, specifies a data-name
	2-3	DDT Memory Load Number
	4-18	DDT ENTRY Address
	19-21	Not used
	22-36	Generator call FISN

Word 3:

Bits	1-36	not used (zero)

Procedure-names

Word 1:

Bits	1-6	Data Analyzer sort key (00)
	7-21	key size = 3
	22-36	item size = 3

Word 2:

Bits	1	= 1, specifies a procedure-name
	2-3	not used
	4-18	beginning FISN of procedure-name
	19-21	not used
	22-36	generator call FISN

Word 3:

Bits	1-36	not used (zero)

4.5.10 Data Name List

1. Table name – Data Name List
2. Table symbol – LN01 set by RUN 1.3
3. Number of words/entry in Index section – 72
   Number of words/entry in Section I-2
   Number of words/entry in Section II – dependent on number of characters in data-name.

4. Number of entries in Sections I and II—dependent on number of data-names in the Data Division Source Program.

5. Table function—used by RUN 8 to produce Data Analyzer listing.

6. Table format—

Index format:

<u>Words 1-72</u>

36 pairs of words for data-names starting with A-Z, 0-9

<u>Odd Word:</u>

Bits	1-12	jump to Section I for data-names with 1-6 characters
	13-24	jump to Section I for data-names with 7-12 characters
	25-36	jump to Section I for data-names with 13-15 characters

<u>Even Word:</u>

Bits	1-12	jump to Section I for data-names with 16-18 characters
	13-24	jump to Section I for data-names with 19-24 characters
	25-36	jump to Section I for data-names with 25-30 characters

Section I

<u>Word 1</u>:

Characters 1-6 of data-name; if data-name < 6 characters master space fill (00)

<u>Word 2</u>:

Bits	1-6	duplication jump
	7-21	Section I location of parent
	22-36	jump to Section II

Section II

<u>Word 1-n</u> remaining characters of data-name n-4

Word n+1

Bits	S	negative
	1-6	file number if file
	7-12	number of parents to unique level of qualification
	13	not used
	14	= 0, not condition switch = 1, condition switch
	15	simple subscript
	16	file $\begin{cases} = 0, \text{ no} \\ = 1, \text{ yes} \end{cases}$
	17	record $\begin{cases} = 0, \text{ no} \\ = 1, \text{ yes} \end{cases}$
	18	condition-name $\begin{cases} = 0, \text{ no} \\ = 1, \text{ yes} \end{cases}$
	19	renamed $\begin{cases} = 0, \text{ no} \\ = 1, \text{ yes} \end{cases}$
	20-21	DDT memory load number
	22-36	DDT entry location

4.5.11 Subroutine CAPS

1. Table name – Subroutine CAPS
2. Table symbol – MS6N set by RUN 6
3. Number of words / entry – 2
4. Number of entries – dependent upon Procedure Division Source Program complexity.
5. Table function – used by RUN 8 in CAP-to-binary conversion and Subroutine Assembly listing.
6. Table format –

   (See 4.6 for a full understanding of this format.)

4.5.12 Running CAPS

1. Table name – Running CAPS
2. Table symbol – MX7N set by RUN 7
3. Number of words/entry -2
4. Number of entries – dependent upon Procedure Division Source Program complexity.
5. Table function – used by RUN 8 in CAP-to-binary conversion and Object Program Assembly listing.
6. Table format –

   (See paragraph 4.6 for a full understanding of this format.)

4.6 MACRO-INSTRUCTIONS AND RELATED TABLE FORMATS

This section gives the final MOBILE I formats of Macro-Instructions, Subroutine Calls, CAP instructions, and some Generated Tables.

4.6.1 Macro-Instructions

4.6.1.1 Macro-Instruction Header Format

Word 1:

Bits 1-6	Macro-instruction key
7-18	Macro-name
19	Transfer
20-21	Q-reg. usage
22-23	not used
24	statement usage
25-30	operand count
31-36	item size

Word 2:

Bits 1-29	FISN
30-36	FISN Macro-counter

In general, the generators turn each Generator Call into a set of macros. If a macro in a set is the destination of a transfer, it must be assigned a unique FISN (an XFISN). Under each FISN, then, there may be a set of macros, all of whick have the same FISN in their headers but each of which has a different FISN Macro-counter, assigned in integer order starting with zero.

The statement bit is set equal to one in the first macro produced for any Generator Call whose statement bit equalled one.

The Q-register usage bits indicate how the Q-register is to be used when this macro is executed. RUN 6 uses this information to minimize the number of loads of the Q-register.

Q-Register Values:

00 = Transparent; no use of Q-register on this macro.

01 = Mask use; exactly one use to load a mask.

10 = Hash use; an arithmetic use which destroys the contents.

11 = Multiple use; last use is a mask.

If the Q-register usage equals 01, the macro expects RUN 6 to create the load. RUN 6 expects the first operand of the macro to be the required mask reference.

The transfer bit is set = 1 if there are any transfers out of this macro, conditional or unconditional. The macro name identifies the macro.

For each macro name there is a unique CAP skeleton store on the Systems tape for RUN 6. A CAP skeleton is simply a block of CAPs which may have any number of blank fields in them to be filled by parameters. The values of the parameters are given to the operands, which occupy one word each at the end of the macro.

4.6.1.2 CAP Modification Words

Between the two-word header and the set of operands, there may be one or more CAP modification words. The CAP modification words control possible deletions and repetitions of CAPs in the CAP skeleton. Every macro has associated with it one of three types of CAP modification, reflected by bits 7-9 in Word 1 of the header:

CAP modification type 0 – no modifications
CAP modification type 1 – repetitions only
CAP modification type 2 – deletions only

In type 0, there are no CAP modification words. In type 1, there is one CAP repetition word for every 36 CAPs in the skeleton. In type 2, there is one CAP deletion word for every 36 CAPs in the skeleton.

In the first repetition and/or deletion word, bits 1 to 36 correspond, in order, to the first 36 CAPs in the skeleton; in the second repetition and/or deletion word, bits 1 to 36 correspond, in order, to the 37th through the 72nd CAPs in the skeleton, etc.

4.6.1.3 Relation of Operands and Skeleton

Bits 13-15 of Word 1 of each CAP in the stored skeleton, called the part number, determine how many parameters in the CAP must be filled with operands, and which positions the parameters occupy in the CAP. Specifically:

Bit 13 = 1 means the "m" or modifier portion must be filled with an operand.

14 = 1 means the "i" or index portion must be filled with an operand.

15 = 1 means the "a" or address portion must be filled with an operand.

13 & 14 = 1 in a "MOV" CAP instruction means the "i-m" or index-modifier portion must be filled with one operand.

The general task of macro-to-CAP conversion performed by Run 6 involves examining each skeleton CAP in order, determining from the part number that n operands are needed to fill the parameters (n = 0, 1, 2, or 3), picking up the next n operands in order from the macro, and placing them properly in the converted CAP. In this fashion all the macro operands are matched with all the CAPs in the skeleton.

Other tasks are performed at the time of macro-to-CAP conversion.

4.6.1.4 Effect of CAP Deletion Word

For any bit in a CAP deletion word which is equal to 1, the corresponding CAP in the skeleton is deleted. The operands which would have filled the CAP if it had not been deleted are now used to fill the next CAP which requires operands.

4.6.1.5 Effect of CAP Repetition Word

For any bit in a CAP repetition word which is equal to 1, the corresponding CAP in the skeleton is to be repeated, each time with different operands (0 to 3 of them) from the macro. The first operand is an integer giving the total number of appearances of the repeated CAP. In the absence of the repetition bit, it would have been inserted in the repeated CAP (or in the next CAP requiring operands if the repeated CAP did not require any). The next operand after the integer operand becomes the first operand in the first appearance of the CAP. Thereafter each set of n operands is placed in one repetition of the CAP until the requested number of appearances has been completed, (where n is the number of operands called for by the part number of the repeated CAP).

4.6.1.6 Macro-Instruction Operands

There are four types of operand in the macros, each occupying exactly one computer word They are: FISN, Source Data, MOBILE Data, and Subroutine Reference

4.6.1.7 Fixed Internal Sequence Number (FISN) Operand

A FISN operand is used for references to locations in the Main Running Program. The format is:

Bit 1	FISN Key (1)
2-30	FISN
31-36	FISN increment

Note that no statement bit is necessary here. A FISN increment of value X has the same effect on the reference as a "+X" has when appended to a symbol in a variable field.

4.6.1.8 Source Data Operand

A Source Data Operand is used for references to fields, records, or buffer areas associated with the input-output files, the Working Storage Section, or the Constant Section of the Source Program Data Division. The format is exactly that of the Data Address Word in the Data Reply, thus permitting the generators to transfer one directly to the other. The format is repeated here:

Bits 1-3	Source Data Key (000)
4-9	File number
10-21	Fixed increment
22-23	Blank
24	Decrement bit
25	Blank
26	Buffer bit
27-36	Final function number

The decrement bit is set equal to 1 if the fixed increment field actually represents a decrement.

The buffer bit is set equal to 1 if the reference is to the buffer area for a file, rather than to a record or field within the file.

The final function number means the final address function serial number assigned by Run 4 in the set of address functions (if there were any) for this reference.

4.6.1.9 MOBILE Data Operand

A MOBILE Data Operand is used for references to tables created by MOBILE and for assembly type references that are also developed internally. The format is:

Bits 1-3	MOBILE Data Key (010)
4-17	Blank
18	Constant bit
19-24	MOBILE Data Symbol
25-36	Increment

The MOBILE Data Symbol is a location symbol for the first address in the table and the increment has the same significance as a "+X" in an Assembly Program, except where the increment is discussed in detail below. The following is a list of the MOBILE Data Symbols currently assigned. (The two octal values listed after the name denote the 6-bit Symbol that goes into bits 19-24 of the MOBILE Data Operand):

File Data (21)—This table includes 16 words of information for each input-output file in the Program. It is used only by the input-output subroutines.

Tape Data (22)—This table includes 1 word for each tape drive used by the Object Program. This table is also used only by the input-output subroutines.

Internal Constant (24)—This table includes all the constants and masks required by the generators.

Alter (25)—This table includes a symbolic transfer for each ALTER statement and each alterable GO statement in the Source Program.

Address Function (27) – This table contains one word for each unique address function reference produced by Run 4 and compressed by Run 6.

Starting Bit (30) – This table contains one word for each unique type 5 address function reference. (Review Section 4. 3. 3.) Such functions involve variable starting bits.

Open Storage (31) – This area serves as temporary storage for the set of macros produced for each Generator Call.

Special Storage (32) – This area includes a set of full-word binary integer fields to which subscript variables, other than simple subscripts, are moved prior to being used in address functions. A simple subscript is defined as a field-word binary integer field in Working Storage.

Undefined Exit (33) – The space for this table is actually reserved in the Initialization area. The nth location in this table, at object time contains the absolute address of the partial entry point ENTRY-N, if the Object Program which includes this partial entry point is loaded at the same time. Otherwise it contains a transfer to a halt and print-out routine. The first location in the Undefined Exit Table is the I-O-BUSY register, which is zero only when the object in-out system has finished processing all in-out instructions stacked in its buffer.

Self-Reference Plus (34) – This Symbol has the same significance as the asterisk (*) in an assembly program.

Self-Reference Minus (34) – This Symbol has the same significance as the asterisk (*) in an assembly program, except that the increment is treated as a decrement; i.e., "-X" instead of "+X".

If the Constant bit equals 1, then bits 19-21 are ignored and bits 22-36 contain a 15-bit binary constant. This constant can be a register refernce (whose first digit is 7), an absolute address, or any other 15-bit constant. Of course, if this constant is used to fill a modifier field in a skeleton CAP, only the last 12 bits will be interpreted. If it is used to fill an index field, only the last 3 bits would be interpreted.

If no operands are required, a blank word 3 must be appended to conform to the rules of Sort Run 5.

4.6.1.10 Subroutine Reference Operand

A Subroutine Reference Operand is used for any reference to a location in a subroutine, whether it is the entrance or any other location. The format is:

Bits 1-3	Subroutine Reference Key (011)
4-12	Blank
13-18	Increment
19-20	Blank
21	Decrement
22-27	Subroutine name
28-30	Subroutine Internal Symbol
31-36	Subroutine counter

Bits 22-36 are the Subroutine Symbol and bits 13-18 serve as a word increment ("+X" in an assembly program) to this Symbol. If bit 21 equals one, bits 13-18 represent a decrement instead of an increment.

The Subroutine Symbol is divided into three parts as follows:

1. Subroutine Name: a unique name for the Subroutine CAP skeleton stored on the systems tape for Run 6. (A CAP skeleton is simply a block of CAPs that may have any number of blank fields in them to be filled by parameters; for every subroutine or macro, Run 6 has a CAP skeleton available.)

2. Subroutine Internal Symbol: The initial location of the subroutine is automatically assigned "000" as its internal symbol. Otherwise, within each subroutine CAP skeleton, up to seven other locations can be assigned internal symbols for reference purposes. Each 15-bit Symbol associated with one subroutine has the same value in bits 31-36 and 22-37; the Symbols are differentiated solely by the value of bits 28-30, which are the Subroutine Internal Symbol in the operand format.

3. Subroutine Counter: If a subroutine has a fixed CAP skeleton, its Subroutine Counter is always zero, and it is produced only once for the object program, regardless of the number of calls on it. If a subroutine has a variable CAP skeleton, either because of blank operands to be filled with parameters, or because of instructions which can be deleted or repeated, that subroutine is "duplicatable." This means that this subroutine appears once in the Object Program for each unique call on it, and there may be up to 63 calls on a duplicatable subroutine. The counter is used to distinguish between the different compiler-initialized versions of the same subroutine in one Object Program.

Note that a subroutine that is entirely initialized at object time by a calling sequence in the Running Program is, from the viewpoint of MOBILE, a fixed subroutine with the counter equal to zero.

4.6.2 <u>Subroutine Calls</u>

4.6.2.1 Subroutine Call Header Format

<u>Word 1</u>:

Bits	1-6	Subroutine Call Key
	7-21	Blank
	22-36	Item size

<u>Word 2</u>:

Bits	1-9	Blank
	10-12	CAP modification type
	13-21	Operand count
	22-27	Subroutine name
	28-30	Initial Subroutine Symbol (000)
	31-36	Subroutine counter

Bits 22-36 of Word 2 have the same significance here as bits 22-36 of the Subroutine Reference operand, except that in the Subroutine Call Header the initial Internal Symbol, with bits 000 in the middle, must be used.

Each generator, particularly the input-output generator, is responsible for maintaining the Subroutine Counter for any duplicatable subroutine. If any duplicatable subroutine can be called by more than one generator, its counter must be in a General Symbol (as opposed to Table Symbol) location during the generation cycle.

Bits 10-12 of Word 2, the CAP modification type, have the same significance as bits 7-9 of Word 1 in the Macro header format, as the following section shows.

4.6.2.2 Subroutine Call Body

The body of a Subroutine Call is constructed exactly the same as the body of a macro. The header is immediately followed by from 0 up to 20 CAP modification words. The actual number is controlled by the CAP modification type and the number of CAPs in the CAP skeleton for this subroutine.

Following the CAP modification words, if there are any, are any number of one-word operands. These operands can be FISN, Source Data, MOBILE Data, or Subroutine Reference operands just as in a macro, except that in Source Data operands the final function number must be zero.

The interpretation of the CAP modification words and the treatment of the operands in doing the subroutine-to-CAP conversion in Run 6 is exactly the same as in the macro-to-CAP conversion in Run 6.

When the subroutine is fixed, which means that it has no CAP modification words and no operands, a blank word 3 must be appended to the Subroutine Call header before the call is placed in the Batch file output of Run 3. This is done because the fixed key in Sort Run 5 includes bits 1-6 of Word 1, all of Word 2, and all of Word 3; so there must be a Word 3 present in every item. In this situation bits 10-21 of Word 2 will be blank.

4.6.3 CAP Instructions

CAP instructions, also known as CAPs, are produced in Run 6 from the subroutine and macro CAP skeletons. The CAPs are counted in Run 7 and are processed into binary relocatable coding and Symbolic Listing in Run 8. The general format of a CAP is:

Word 1:

Bits	1-3	CAP type
	4-9	Machine instruction or pseudo-op
	10	Not used
	11	Address function
	12	Statement
	13-14	Blank
	15	Decrement
	16-36	Address portion

Word 2:

Bits 1-15	CAP symbol
16-36	Index-modifier portion

4.6.3.1 CAP Symbols

Except for the CAP Symbol, the fields in the format have the same significance for subroutine CAPs as for Main Program Running CAPs (or just Running CAPs).

In a subroutine CAP, the CAP Symbol is a Subroutine Symbol, consisting of the three fields, Subroutine Name, Internal Symbol, and Subroutine Counter just as in bits 22-36 of a Subroutine Reference Operand.

In a Running CAP, the CAP Symbol is a NISN (ALL FISNs are converted to NISNs in Run 6).

4.6.3.2 Other Fields in CAP Format

There are two CAP types at present. Type 0 is the ordinary assembly machine instruction. Type 1 is either an assembly pseudo-operation, such as BSS, or a Special CAP. The Special CAPs to date are the two CAPs involved in ALTER statements; the Alterable TRU CAP and the Alter MOV CAP.

For all type 0 CAPs, bits 4-9 contain the actual machine instruction, that is, the 6 bits that go into bits 1-6 of the binary instruction.

For types 1 and 2 CAPs, bits 4-9 contain a code corresponding to the particular pseudo-op or special CAP.

The Address Function bit equals 1 in any CAP which was inserted by Run 6, not as a result of processing a macro CAP skeleton, but for computing the loading address functions.

The purpose of the Address Function bit is to cause Run 8 to place some special English description on the same line in the Triple Listing of this instruction. The statement bit equals 1 in the first CAP in any CAP skeleton of a macro whose statement bit equalled 1.

The decrement bit equals 1 if the following conditions were met in Run 6:

1. A source Data operand was inserted into the address portion of this CAP when its skeleton was processed.
2. The decrement bit (bit 15) equalled 1 in the Source Data Operand reference.

4.6.3.3 Address Portion Field in the CAP Format

There are four possible operand types in the Address Portion field, and these correspond exactly to the four types of operand references in the body of a Macro-instruction or Subroutine Call. The formats of the address operands are as follows:

Bits	16-17	Source Data Operand Key (00)
	18	Buffer
	19-24	File number
	25-36	Fixed increment
	16-17	MOBILE Data Operand Key (01)
	18	Constant
	19-24	MOBILE Data Symbol
	25-36	Increment
	16-17	Subroutine Reference Operand Key (10)
	18-20	Blank
	21	Decrement
	22-36	Subroutine Symbol
	16-17	NISN Operand Key (11)
	18-20	Blank
	21	Decrement
	22-36	NISN

The decrement bit in this Subroutine Reference operand is the same as the decrement bit (bit 21) in the Subroutine Reference operand in a Macro or Subroutine Call. The Decrement bit in the NISN operand cannot come from a FISN operand reference in a macro or Subroutine Call. It can only appear as a result of manipulations by Run 6 for optimization purposes. For both Subroutine Reference and NISN operands the 6-bit increment which comes from the Macro or Subroutine Call operand is placed in bits 16-21 of the index-modifier portion field in Word 2 of the CAP format. It is understood that usually the decrement bit will be 0 meaning that the increment is positive, and usually the increment itself will be zero meaning that there is no increment.

4.6.3.4 Index-Modifier Portion Field in the CAP Format

There are three possible interpretations of index-modifier portion field. The interpretations do not depend on key bit as in the address portion field, but they do depend on the CAP instruction. The formats are as follows:

1. For any input-output instruction:

Bits 16-18	Blank
19-24	Selected Device Code or "J" portion
25-27	Blank
28-36	Word-block count or "K" portion

2. For the MOV instruction, the index-modifier portion field is interpreted as a single symbolic address in the same manner as the address portion field. The only difference is that there can be no increment or decrement for Subroutine Reference or NISN operands. Although not relevant to Runs 7 and 8, it should be pointed out that no Source Data operands with non-zero final function numbers can be inserted in a MOV skeleton CAP because of the absence of indexing.

3. For all other instructions

Bits 16-21	Increment for Subroutine Reference or NISN operand in Address portion field
22-24	Index, or "G" portion
25-36	Modifier, or "B" portion

4.6.3.5 Special CAP Formats

1. Format of the Alterable TRU CAP:

<u>Word 1</u>:

Bits	1-3	CAP type (1_8)
	4-9	Machine instruction (40_8)
	10-12	Same as ordinary CAP
	13-14	Blank
	15	Decrement
	16-36	Address portion: Subroutine Reference or NISN operand as in ordinary CAP

<u>Word 2</u>:

Bits	1-15	CAP Symbol
	16-21	Increment for Address portion operand
	22-26	Blank
	27-36	New Serial number i

2. Format of the Alter MOV CAP:

<u>Word 1</u>:

Bits	1-3	CAP type (1_8)
	4-9	Machine instruction (52_8)
	10-12	Same as Ordinary CAP
	13-21	Destination Serial number i
	22-36	NISN or subroutine symbol

<u>Word 2</u>:

Bits	1-15	CAP Symbol
	16-24	New Serial number j
	25-26	Blank
	27	Subroutine
	28-36	Origin Serial number k

If the Subroutine bit in the Alter MOV CAP (bit 27 of Word 2) equals 1, then bits 22-36 of Word 1 are to be interpreted as a Subroutine Symbol; otherwise, they are interpreted as a NISN.

4.6.4 Object Core Allocation for the Listing

The core storage area, during the run of a single Object Program (as opposed to a set of group-loaded programs) is divided into six major areas for listing purposes:

1. Initialization—This includes some service routines, such as the Rerun Stop and Restart Routines, the Partial Entry Routine and Intersegment Monitor, and the Undefined Exit and Trap Tables where required. The routines in this area are initially loaded by the MOBILE loader.

2. Input-Output File—This includes all the single and double buffer areas and current record areas assigned to the input and output files named in the Source Program. No data is entered into this area by the MOBILE loader. All the areas after the Input-Output File Area are filled with the Object Program and required data by the MOBILE loader.

3. Working Storage—This includes the core space necessary to contain all the records and fields named in the Working Storage Section.

4. MOBILE Data—This area includes all the MOBILE Data Tables, such as File Data, Tape Data, Internal Constant, Alter, Address Function, Starting Bit, Open Storage, and Special Storage. It also includes the Transfer Table, if one is needed, and the External Constants which the user requested in his Constant Section. Note that although Working Storage and Constant data are considered to be files number 1 and 2 throughout compilation, for purposes of core allocation and the listing, they are not included in the Input-Output File Area.

5. Subroutine—This area includes the binary instructions of the fixed and duplicatable subroutines requested by the generators.

6. Main Running Program—This area includes the binary instructions resulting from direct translation of the Procedure Division of the Source Program. Any USE or ENTER Declarative coding comes first, followed by the coding for the Main Body of the Procedure Division. The first line of coding for the Main Body is considered to be the entrance point of the Object Program.

For purposes of relocation of binary instruction operands, the last three of these Major Areas are combined into one area called the Running Program Area.

4.6.5 Generated Table Formats

4.6.5.1 Alter Table Format

Word 1:

Bits	1-6	Alter Key
	7-21	Serial number
	22-30	Blank
	31-36	Item Size (00003_8)

Word 2:

Bits	1-29	Destination FISN
	30-35	Blank
	36	Alter bit

Word 3:

Bits	1-29	Origin FISN
	30-36	Blank

4.6.5.2 XFISN Table Format

Word 1:

Bits	1-6	XFISN Table Key
	7-30	Blank
	31-36	Item size (3)

Word 2:

Bits	1-29	XFISN
	30-36	Blank

Word 3:

Bits	1-36	Blank

4.6.5.3 Link Table Format

<u>Word 1:</u>

Bits	1-6	Link Table Key
	7-9	High Frequency
	10-12	Frequency Assigned
	13-15	Perform Range Enter
	16-18	Perform Range Exit
	19-21	Single Exit
	22-24	Missing Fall Through Bits
	25-30	Blank
	31-36	Item Size

<u>Word 2:</u>

Bits	1-29	Destination FISN
	30-36	Blank

<u>Word 3:</u>

Bits	1-29	Origin FISN
	30-36	Blank

No Link Table entry is to be produced for the fall-through associated with conditional transfer.

Sort Run 5 uses bits 1-6 of Word 1 as the major key, Word 2 as the intermediate key, and Word 3 as the minor key. Bits 31-36 of Word 1 contains the item size.

4.7 MOBILE COMPILER OUTPUT LISTINGS

This section describes the format of the 5 types of output listings that appear on the Compiler Printing Tape (tape - 43). On-line printing of all listings will occur on the line printer unless the user wishes to suppress certain listings.

MOBILE produces five types of output listings; these are:

1. Block Source Program
2. Core Storage Allocation
3. Data Analyzer
4. Subroutine Assembly
5. Object Program Assembly (also referred to as a Triple Listing)

Each listing is produced and written on the Compiler Printing Tape (tape - 43) in the above order with an EOF separating each output listing. Each listing that is not suppressed by the programmer is listed on the on-line printer at compilation time. Since every listing is on tape, it is possible to obtain a particular suppressed listing at a later time (assuming the listing tape has been saved).

In order to suppress a particular output listing from being listed on the on-line printer, the word NO must be punched in the correct columns of the control card, which is the first card of every compilation as listed below:

Listing	Card Columns	
1. Block Source Program	38-39	No = Suppress; Blank = On-line Printer
2. Core Storage Allocation	40-41	No = Suppress; Blank = On-line Printer
3. Data Analyzer	44-45	No = Suppress; Blank = On-line Printer
4. Subroutine Assembly	46-47	No = Suppress; Blank = On-line Printer
5. Object Program Assembly (triple)	42-43	No = Suppress; Blank = On-line Printer

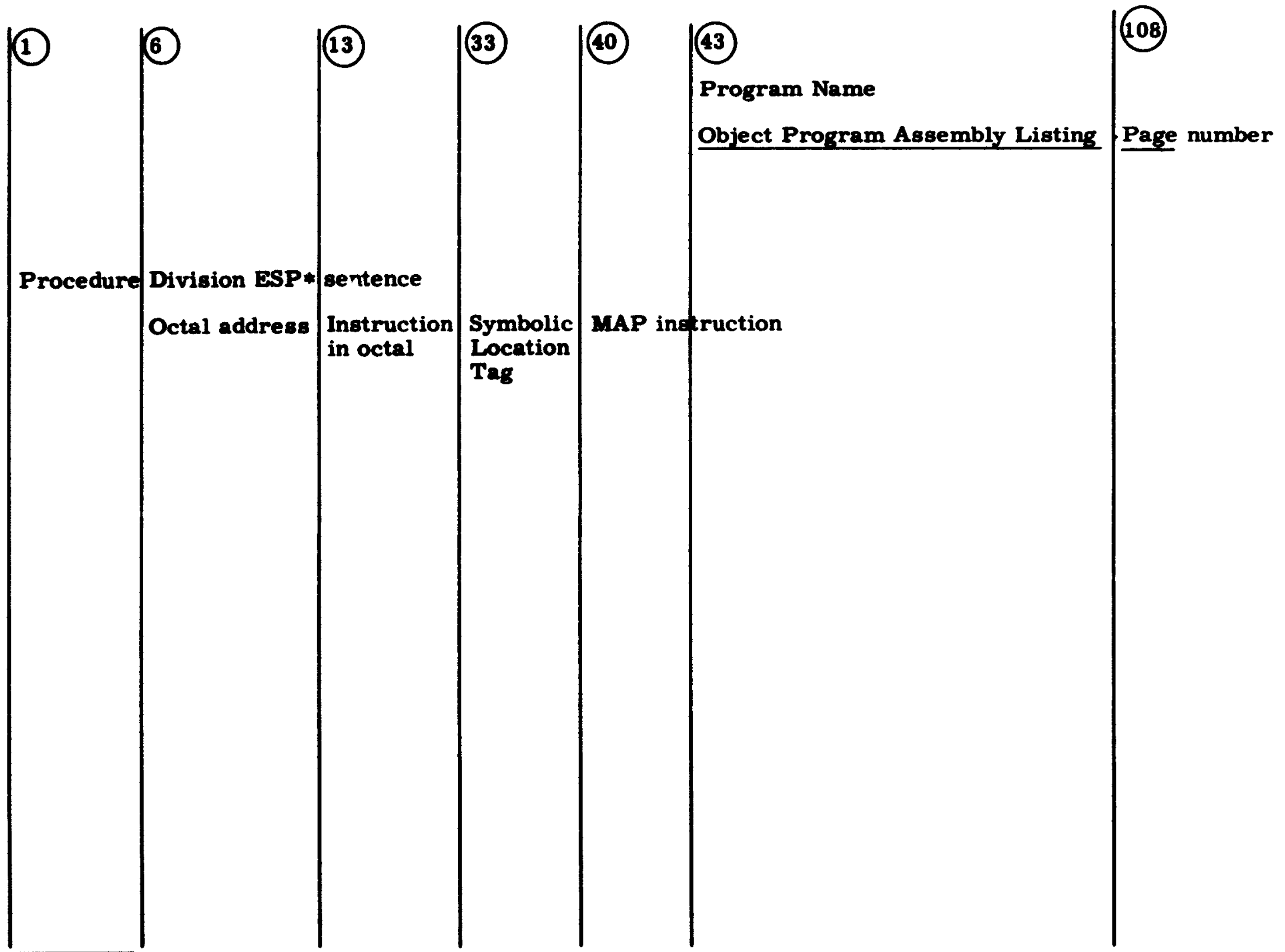
1	8	43	87	108	115
		Program name			
		<u>Block Source Program Listing</u>		<u>Page</u>	Number
Card Sequence number, if present on input	One line	of Source Program	Symbolic ERROR flags		Program identification, if present on input

(13)	(43)	(49)	(67)	(85)	(108)
	Program	name			
	Core Storage Allocation Listing				Page number
		A. General Core Areas			
Area name	Symbol		starting Octal address	Decimal word size	
Initialization	A				
Input-Output Files	D				
Working Storage	E				
External Constants	F				
MOBILE Data Storage	K				
File Data Table	L				
Tape Data Table	M				
Multiple File Reel Data	N				
Internal Constants	O				
Alter Table	P				
Address Function Values	R				
Address FCN Starting Bits	S				
Open Storage	T				
Special Storage	U				
Undefined Exit	V				
Subroutine Instructions	)				
Running Program Instructions	*				
Running Program End	0				
		B. File areas			
File name	File number		starting Octal address	Decimal word size	
		C. Subroutine areas			
Subroutine name	Symbol		starting Octal address		

(1)	(37)	(43)	(67)	(79)	(108)
		Program name			
		Data Analyzer Listing			Page number
		A. Data-names			
Place of reference*	Data-name			Qualifiers**	
		B. Procedure-names			
Place of reference*		NISN***	Procedure-name		

*Paragraph or Section-name
**Parent data-names used in qualification of a data-name
***New internal sequence number

6	13	33	40	43	108
				Program name	
				<u>Subroutine Assembly Listing</u>	<u>Page</u> number
Octal address	Instruction in octal	Symbolic Location Tag	One	MAP instruction	



*Expanded Source Program

4.8 QUALIFICATION TASK IN RUN 1.3 AND DATA NAME LIST DESCRIPTION

4.8.1 Introduction

If a data-name "A" has been used to denote more than one unit of data, "A" must still be made unique. Uniqueness of a data unit can be obtained by the Qualification Method, i.e., "A" which is contained in B which is contained in C. . . . etc. Up to 49 levels of Qualification may be used (including the file level). Since the data-name, referring to the unit of data, (base-data-name) may not be unique in itself, it is qualified by other data names that have lower level numbers and therefore higher rank in the hierarchy of data-names until uniqueness is achieved. Unnecessary qualification is allowed; i.e., qualification beyond uniqueness. For instance, for purposes of documentation, unnecessary qualification may be helpful. Direct ascent through the hierarchy of data-names is not necessary; that is, a base data-name may be made unique by qualification in a group item, record or file, skipping all intermediate levels of qualification. The section of Run 1.3 which traces this qualification and thus obtains the unique base data-name if it exists is called "PT 166".

4.8.2 Table Input Descriptions

Input to "PT 166" is in the form of Tables, DNLA and DNMLA. DNLA is the Data Name List. DNMLA refers to the entries in DNLA of the base data name and its qualifiers. DNMLA entries are not the exact entries in DNLA of the above-mentioned data-names. They are merely the first of all such data-names; i.e., the base-data-name "A" will have as its entry in DNMLA a reference to the first "A" of perhaps several "A"'s in DNLA. It is the task of "PT 166" to determine which "A", if any, meets the requirements of having all the qualifiers listed in DNMLA + 1, +2, +N. The Data Name List is divided into three parts: Index, Section I, and Section III.

The index consists of two-word entries, one entry for each of the letters of the alphabet and one entry for each of the numbers 0 - 9. Therefore, the index is 72 words in length. Each of these two-word entries consists of six 12-bit jump numbers relative to the start of the DNLA*. These jump numbers are determined

*The first word of the DNLA immediately preceding the index is called the block label word; jumps are relative to this.

by the number of characters in a data-name. If a data-name (containing N characters) begins with a letter or number "X", then one obtains from the index entry for "X" and the jump number for "N" characters the Section I entry for the first of all data-names starting with "X" and having "N" characters.

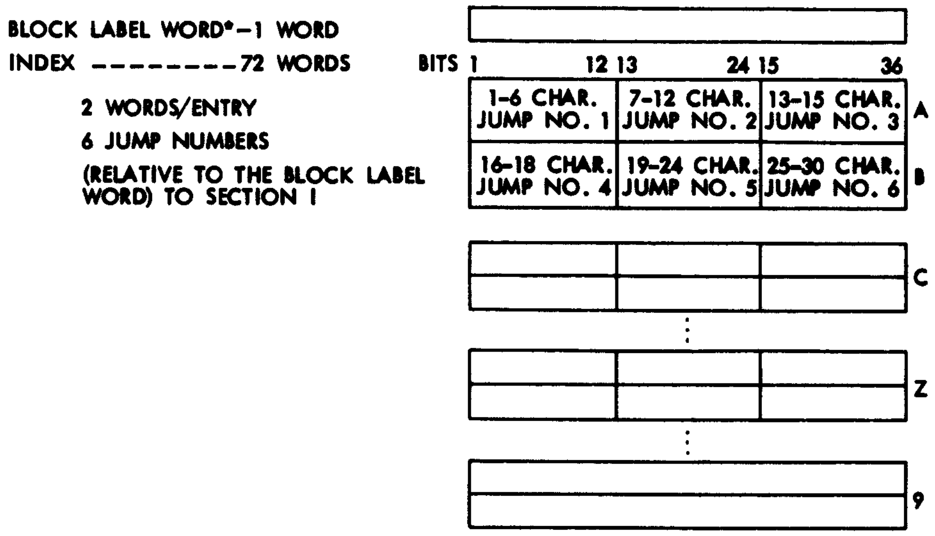
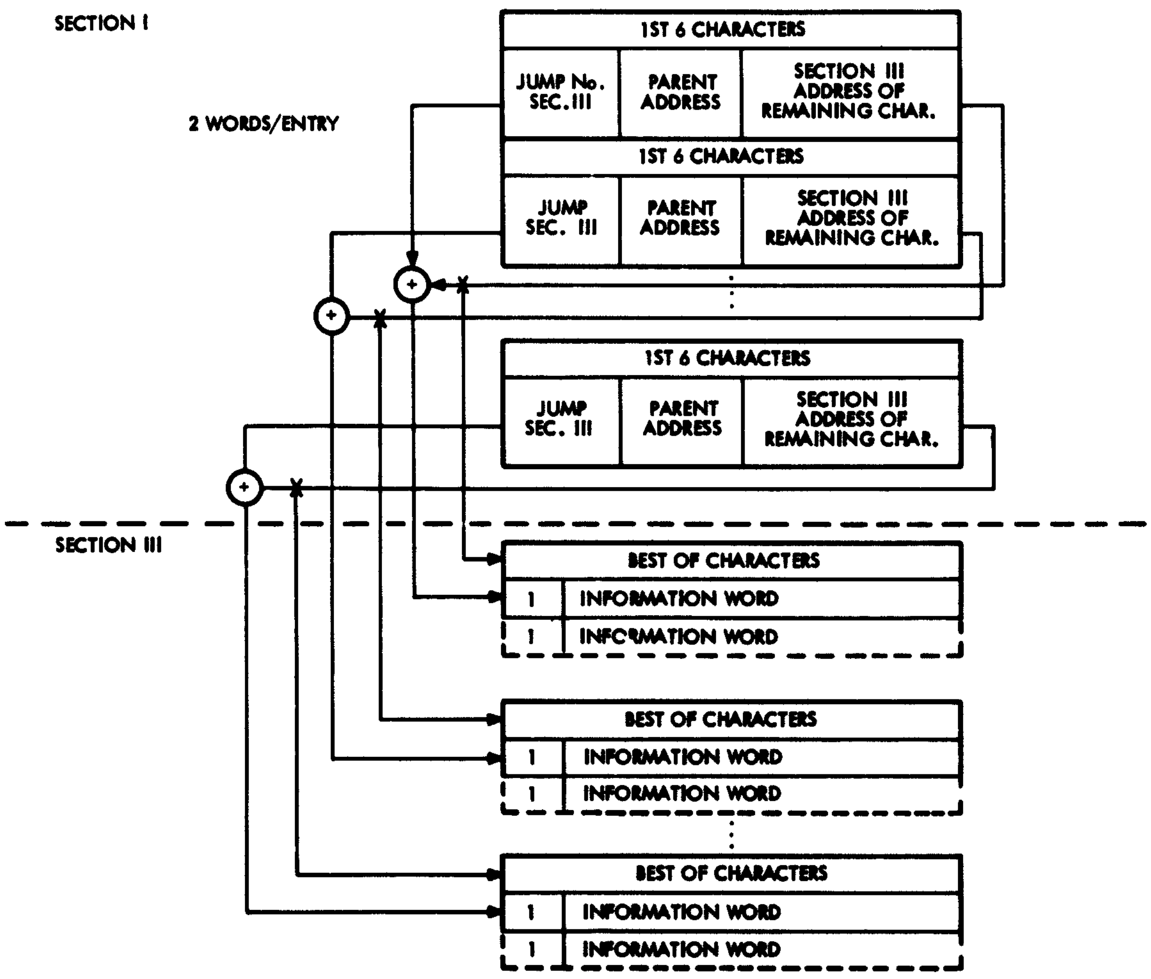


Figure 4-1. Structure of Index for Data-Name List

The Section I entry (two words/entry) for a data-name consists of the first six characters of the data-name (1st word) and a second word which is divided into three parts. Part 1, operation code field, is a 6-bit jump number. Part 2, g-b field, is a 15-bit absolute address of the Section I entry of the parent** of this data-name. Part 3, the address field, is a 15-bit absolute address of the remaining characters of the data-name if any exists (Section III entry). When the Part 1 value is added to the Part 3 value, the result is an absolute address for the information word*** (Section III entry) of the data-name being considered.

*The first word of the DNLA immediately preceding the index is called the block label word; jumps are relative to this.
**That unit of data which immediately contains this data-name; i.e., the first parent as opposed to the first qualifier.
***DDT Reference Word Address



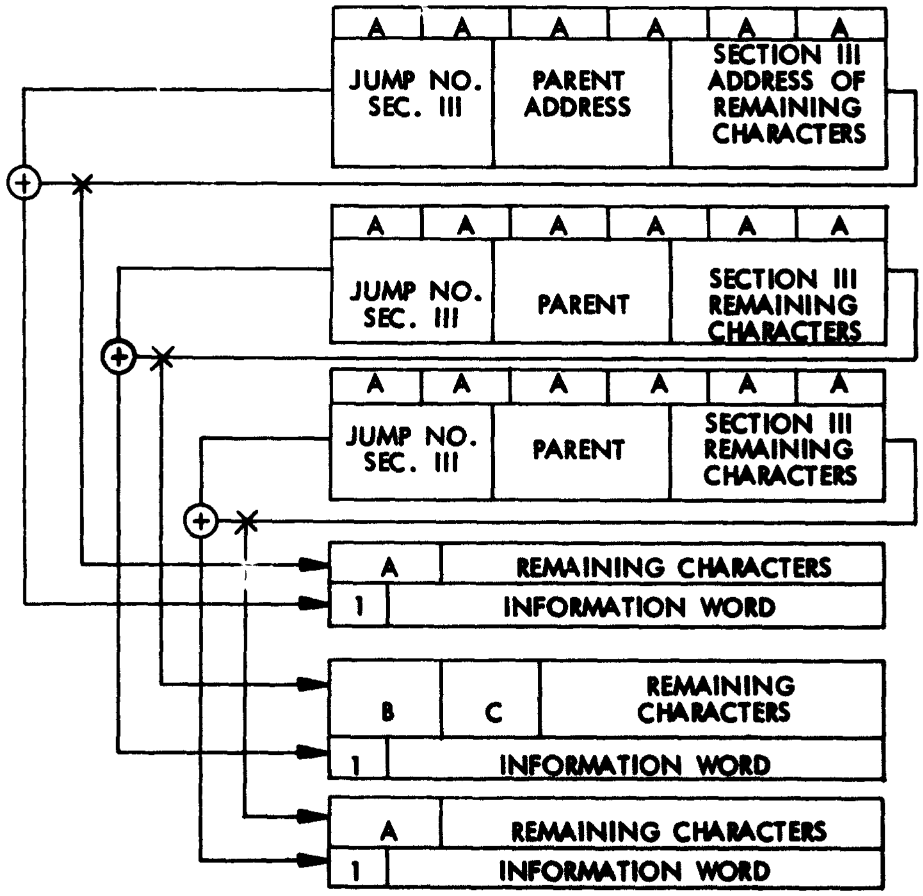
NOTE: + IMPLIES ADDITION.
X IMPLIES THAT THIS INFORMATION PASSES IN TWO DIRECTIONS AS INDICATED BY THE LINES.
- - - - - - - - - -IMPLIES THERE MAY BE MORE THAN ONE INFORMATION WORD IN A GIVEN SECTION III ENTRY IF THERE ARE TWO OR MORE EQUAL BASE DATA NAMES. THIS IS NOT THE CASE AS DEPICTED ABOVE.
1 IMPLIES THAT INFORMATION WORD IS NEGATIVE.

41-60

Figure 4-2. Structure of Section I and Section III

The Section III entry for a data-name contains the remaining characters of the data-name (if any) and an information word for each of the data-names which have this Section III entry as their Section III entry.

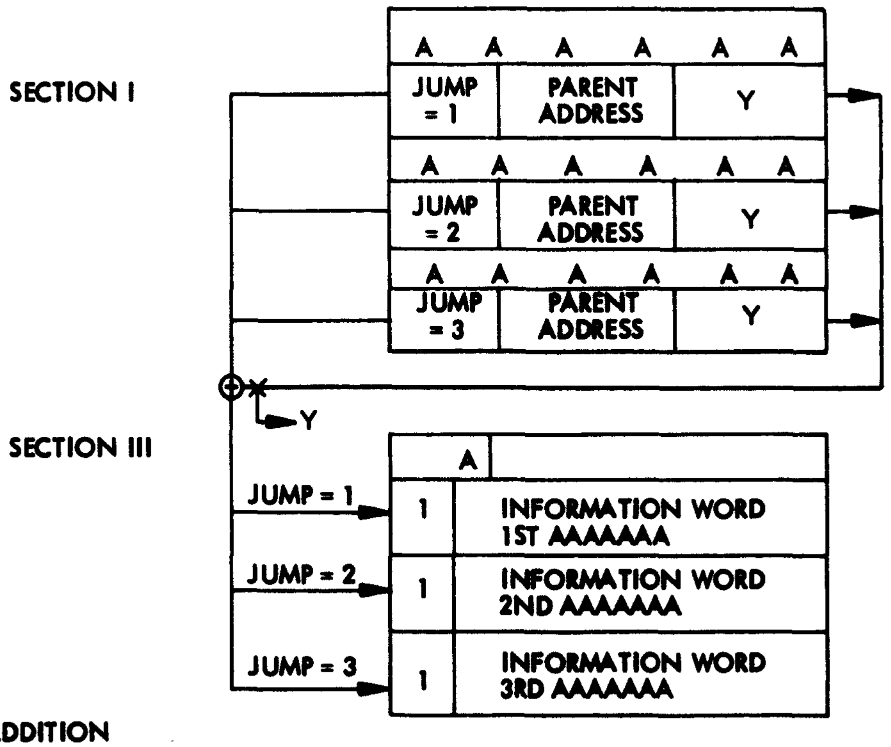
The Data Name List is sorted alphabetically on the first word and numerically on the 6 possible size groups to which the complete data-name belongs (see page 4-76). Therefore, it is possible for a situation as outlined below to occur when dealing with duplicated data-names. For example, consider the data-name, AAAAAAA, presented below:



NOTE: + IMPLIES ADDITION. X IMPLIES THAT THIS INFORMATION PASSES IN TWO DIRECTIONS AS INDICATED BY THE LINES.

Figure 4-3. Possible Structure of DNLA Containing Duplicated Data-Names

Besides the previous possible structure of DNLA containing duplicated data-names, the following possible structure might also occur:



NOTE:
+ IMPLIES ADDITION
X IMPLIES THAT THE INFORMATION PASSES IN TWO DIRECTIONS AS INDICATED BY THE LINES.

Figure 4-4. Other Possible Structure of DNLA Containing Duplicated Data-Names

A DNLA with duplicated data-names may appear in either of the two structures outlined or in any combination of them.

DNMLA is the other table used by "PT 166". As was mentioned before, it contains information regarding qualification. The first word of DNMLA contains in its g-b field a 15-bit absolute address of the Section III entry for this particular base data-name. In the address field of this first word is the Section I absolute address. This is not necessarily the address of the wanted base data-name, but merely the first such entry. In DNMLA +1 is a field with similar information, but this time it refers to the first qualifier (least inclusive). In DNMLA + 2 is the information concerning the next qualifier. In DNMLA + N is the information concerning the last qualifier.

	S	OP	I-M	A	
DNMLA	0		SECTION III ADDRESS	SECTION I ADDRESS	BASE DATA NAME
DNMLA + 1	0		SECTION III ADDRESS	SECTION I ADDRESS	1ST QUALIFIER
DNMLA + 2	0		SECTION III ADDRESS	SECTION I ADDRESS	2ND QUALIFIER
DNMLA + N*	1	SECTION III ADDRESS		SECTION I ADDRESS	NTH QUALIFIER

Figure 4-5. Structure of DNMLA

4.8.3 Procedure

"PT 166" begins with the first base data-name (pointed out in DNMLA) and traces its ancestry to see if its parents, grandparents, etc., match the qualifiers in DNMLA + 1, etc. If the first base data-name does not meet these requirements, DNLA is searched until the area in which such base data-names may be found is exhausted. The same process described above is carried out with each base data-name. If a match is found, the search continues to determine that no other base data-name has met the same requirements to insure uniqueness. A count is kept of the number of levels of qualification encountered (by the parent, grandparent, etc., method) since the qualifiers need not be in immediate succession in DNMLA. ** This count is passed on for use by Run 8.

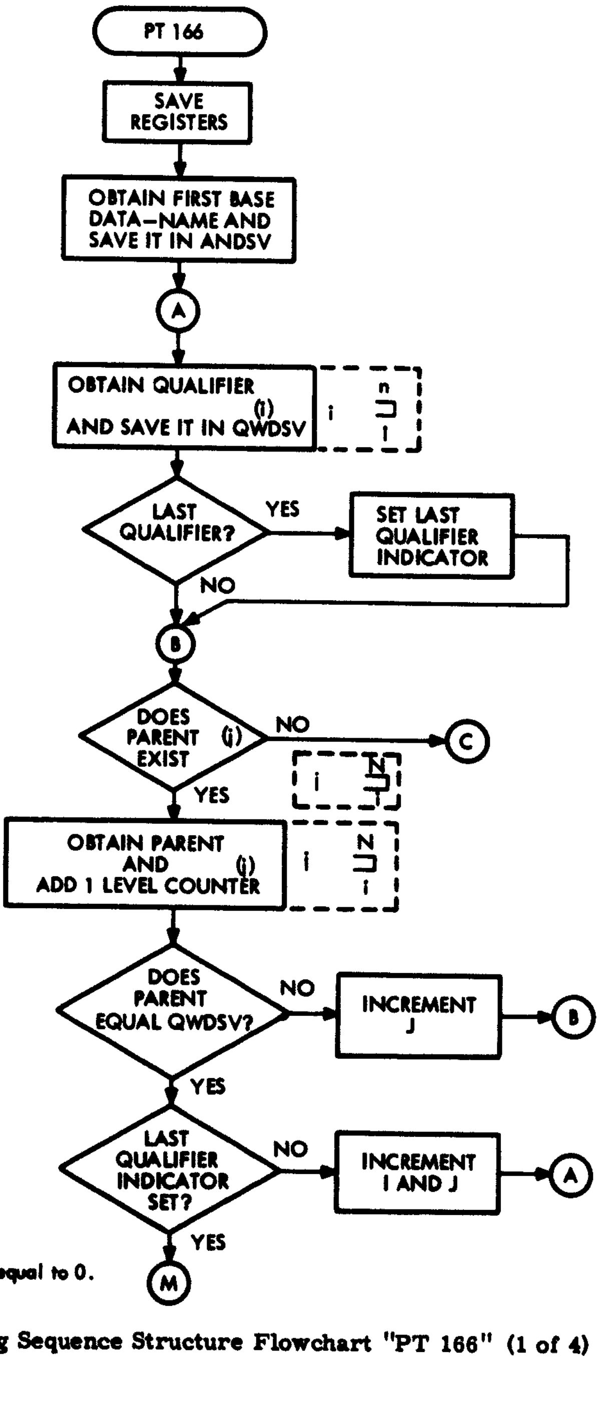
*Negative sign indicates the last (most inclusive qualifier).

**A may be contained in B, C, D, but may be qualified as A in (of) D if this is sufficient.

CALLING SEQUENCE STRUCTURE "PT 166"

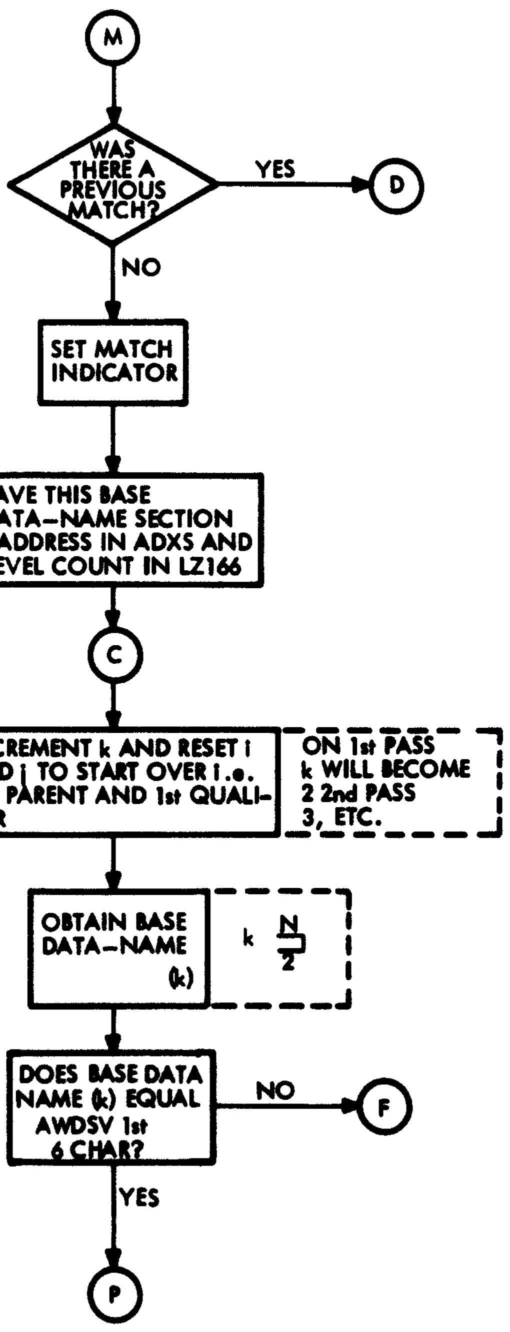
A	TRL	PT 166	
A + 1			Error 1 return: multiple-defined. Return here if multiple-defined with Section I Word 1 address of first match in ACC and level count in QRG.
A + 2			Error 2 return: undefined. Return here if undefined with zero in ACC and QRG.
A + 3			Normal return: Address of Section I Word 1 in ACC and level count in QRG.

NOTE: This routine assumes absolute address of Section I of data-names and qualifiers in DNMLA (address field). It further assumes the last entry in DNMLA, i.e., the last qualifier, will have a negative sign bit.


PT 166
SAVE REGISTERS
OBTAIN FIRST BASE DATA—NAME AND SAVE IT IN ANDSV
A
OBTAIN QUALIFIER (i) AND SAVE IT IN QWDSV
i
n ⊔ i
LAST QUALIFIER?
YES
SET LAST QUALIFIER INDICATOR
NO
B
DOES PARENT (j) EXIST
NO
C
i
N ⊔ j
YES
OBTAIN PARENT AND (j) ADD 1 LEVEL COUNTER
i
N ⊔ j
DOES PARENT EQUAL QWDSV?
NO
INCREMENT J
B
YES
LAST QUALIFIER INDICATOR SET?
NO
INCREMENT I AND J
A
YES
M
22-43

*I.e., is parent address equal to 0.

Figure 4-6. Calling Sequence Structure Flowchart "PT 166" (1 of 4)


M
WAS THERE A PREVIOUS MATCH?
YES
D
NO
SET MATCH INDICATOR
SAVE THIS BASE DATA-NAME SECTION I ADDRESS IN ADXS AND LEVEL COUNT IN LZ166
C
INCREMENT k AND RESET i AND j TO START OVER i.e. 1ST PARENT AND 1st QUALIFIER
ON 1st PASS k WILL BECOME 2 2nd PASS 3, ETC.
OBTAIN BASE DATA-NAME (k)
k N/2
DOES BASE DATA NAME (k) EQUAL AWDSV 1st 6 CHAR?
NO
F
YES
P

22-43

Figure 4-6. Calling Sequence Structure Flowchart "PT 166" (2 of 4)

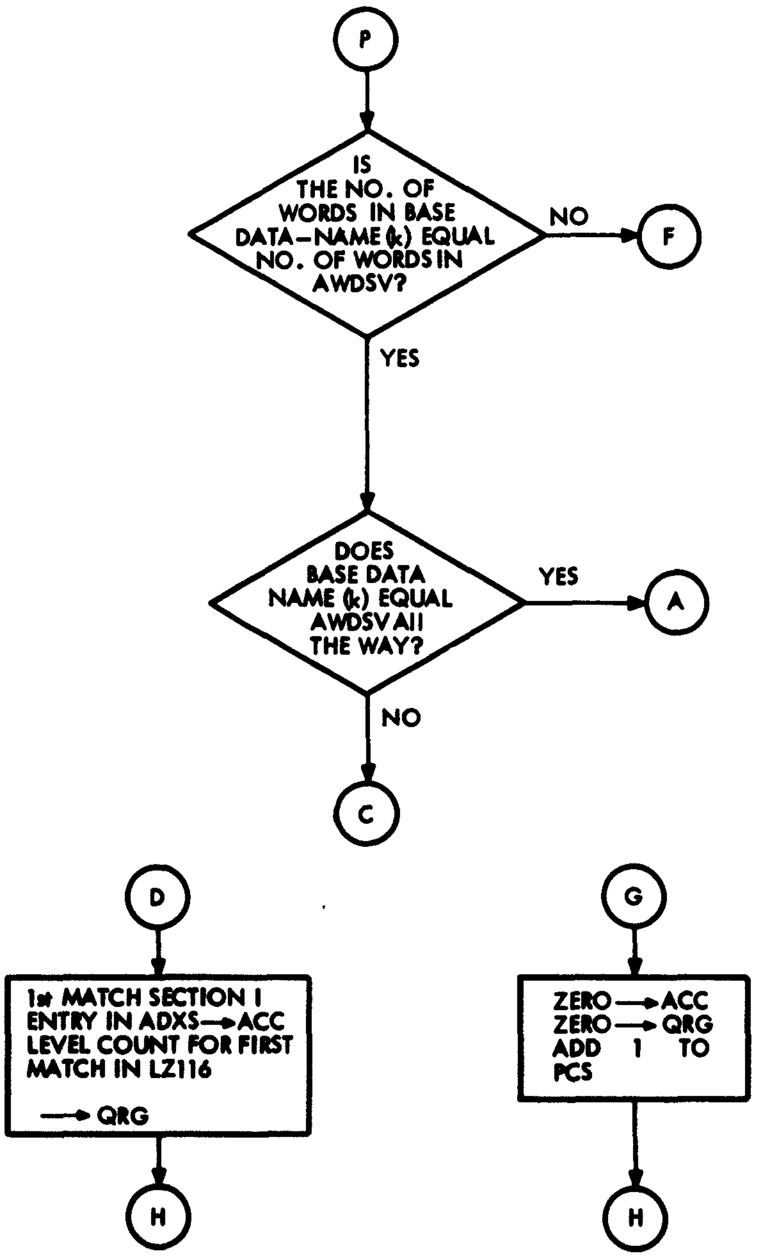

P
IS THE NO. OF WORDS IN BASE DATA-NAME (k) EQUAL NO. OF WORDS IN AWDSV?
NO
F
YES
DOES BASE DATA NAME (k) EQUAL AWDSV All THE WAY?
YES
A
NO
C
D
1st MATCH SECTION I ENTRY IN ADXS→ACC LEVEL COUNT FOR FIRST MATCH IN LZ116 → QRG
H
G
ZERO→ACC
ZERO→QRG
ADD 1 TO PCS
H
34-63

Figure 4-6. Calling Sequence Structure Flowchart "PT 166" (3 of 4)

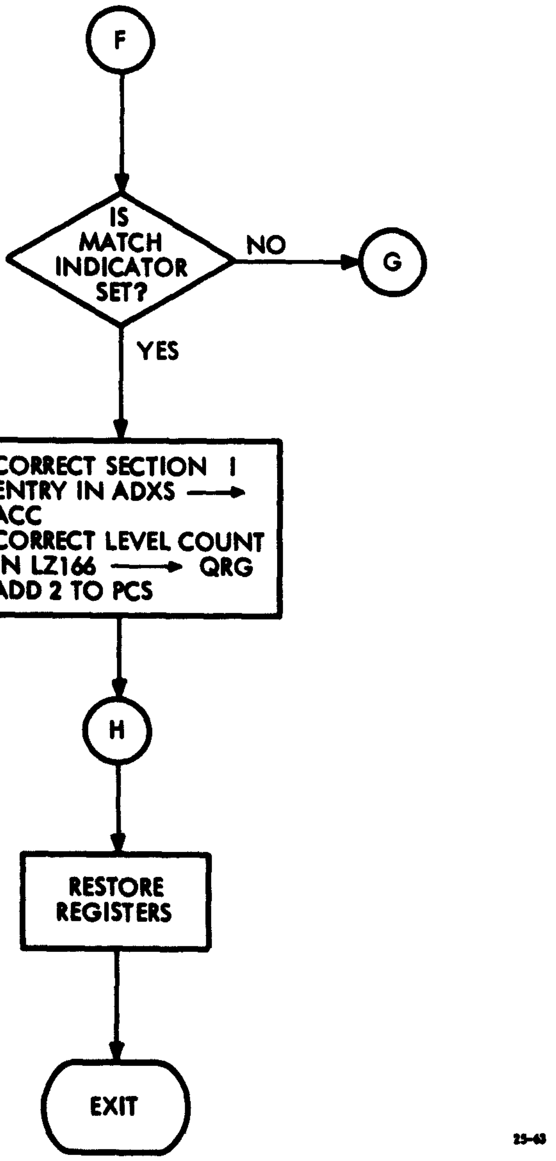

F
IS MATCH INDICATOR SET?
NO
G
YES
CORRECT SECTION I ENTRY IN ADXS → ACC
CORRECT LEVEL COUNT IN LZ166 → QRG
ADD 2 TO PCS
H
RESTORE REGISTERS
EXIT
25-63

Figure 4-6. Calling Sequence Structure Flowchart "PT 166" (4 of 4)

SECTION V

CONCLUSIONS

MOBIDIC MOBILE I is complete and is undergoing system testing. Trouble areas are being crystallized during the testing, and errors are being corrected, thus brining MOBIDIC MOBILE I to full operational status.

SECTION VI

PROGRAM FOR THE NEXT PERIOD

During the next period acceptance testing will be completed. Final documentation, including a Final COBOL Reference Manual, will be published.

SECTION VII

IDENTIFICATION OF KEY PERSONNEL

7.1 KEY TECHNICAL PERSONNEL

Alvin H. Hatch	Manager, Applied Programming Department
Arthur S. Morse	Section Head, Language Implementation Section
Herbert S. Hughes	Research Engineer
Roy Sundgren	Senior Engineer
Richard Mackler	Senior Engineer

7.2 APPROXIMATE MAN-HOURS EXPENDED

Name	Hours
Alvin H. Hatch	102
Arthur S. Morse	511
Herbert S. Hughes	549
Roy Sundgren	475
Richard Mackler	524
Total	2161

DISTRIBUTION LIST

THIRD QUARTERLY REPORT

DA-36-039-sc-89231

	No. Copies
OASD (R&E) Rm 3E1065 ATTN: Technical Library The Pentagon Washington 25, D.C.	1
Chief of Research and Development OCS, Department of the Army Washington 25, D.C.	1
Chief Signal Officer ATTN: SIGRD Department of the Army Washington 25, D.C.	1
Director, U.S. Naval Research Laboratory ATTN: Code 2027 Department of the Army Washington 25, D.C.	1
Commanding Officer and Director U.S. Navy Electronics Laboratory San Diego 52, California	1
Commander Aeronautical Systems Division ATTN: ASPRDL Wright-Patterson Air Force Base, Ohio	1
Commander, Rome Air Development Center ATTN: RAALD Griffiss Air Force Base, New York	1
Commanding General U.S. Army Electronic Proving Ground ATTN: Technical Library Fort Huachuca, Arizona	1
Commanding General U.S. Army Electronic Proving Ground ATTN: ADP Department Fort Huachuca, Arizona	1

	No. Copies
Commander, Armed Services Technical Information Agency ATTN: TIPCR Arlington Hall Station Arlington 12, Virginia	10
Chief, U.S. Army Security Arlington Hall Station Arlington 12, Virginia	2
Deputy President U.S. Army Security Agency Board Arlington Hall Station Arlington 12, Virginia	1
Commanding Officer U.S. Army Electronic Material Support Agency ATTN: SELMS-ADJ Fort Monmouth, New Jersey	1
Corps of Engineers Liaison Office U.S. Army Electronic Research & Development Laboratory Fort Monmouth, New Jersey	1
Commanding Officer U.S. Army Electronic Research & Development Laboratory Logistics Division ATTN: Mr. N.J. Taupeka, SELRA/NPE Fort Monmouth, New Jersey	3
Commanding Officer U.S. Army Electronic Research & Development Laboratory Data Equipment Branch, Data Processing Facilities Div. Fort Monmouth, New Jersey	1
Commanding Officer U.S. Army Electronic Research & Development Laboratory ATTN: Director of Engineering Fort Monmouth, New Jersey	1
Commanding Officer U.S. Army Electronic Research & Development Laboratory ATTN: Technical Documents Center Fort Monmouth, New Jersey	1
Commanding Officer U.S. Army Electronic Research & Development Laboratory ATTN: Technical Information Div. Fort Monmouth, New Jersey	3

	No. Copies
ADPS Committee Officer's Department U.S. Army Signal School Fort Monmouth, New Jersey	1
Information Processing Branch Dept. of Specialist Training U.S. Army Signal School Fort Monmouth, New Jersey	1
MOBIDIC Project Office CCIS-70 PMSO U.S.A. Electronics Command Fort Monmouth, New Jersey	1
Ordnance Stock Control Agency APO 58 ATTN: Maj. C.S. Moody New York, New York	1
7th Army Stock Control Center APO 872 ATTN: Capt. D. Lasher New York, New York	1
Philco Corporation ATTN: Mr. S. Berkowitz 3900 Welsh Road Willow Grove, Pa.	1
RCA Surface Comm. Division ATTN: Mr. A. Coleman Camden, New Jersey	1